

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»  
Факультет інформатики  
Кафедра мультимедійних систем

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ РЕПЕТИТОРІВ

**Текстова частина до курсової роботи**

**За спеціальністю 121 «Інженерія програмного забезпечення»**

Керівник курсової роботи

ст.в. Борозенний С.О.

*(прізвище та ініціали)*

\_\_\_\_\_

*(підпис)*

“ \_\_ ” \_\_\_\_\_ 2021 р.

Виконав студент Копійка В.Г.

*(прізвище та ініціали)*

\_\_\_\_\_

*(підпис)*

“ \_\_ ” \_\_\_\_\_ 2021 р.

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»  
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри  
мультимедійних систем,  
доцент, к.ф-м.н.  
\_\_\_\_\_ О.П. Жежерун  
(підпис)  
“ \_\_\_\_ ” \_\_\_\_\_ 2020 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу  
студенту Копійці Вадиму Геннадійовичу  
Факультету інформатики 3 р.н. бакалаврської програми  
ТЕМА: Розробка мобільного застосунку для репетиторів

Зміст ТЧ до курсової роботи:

Анотація

Вступ

Аналіз предметної області, теоретичні відомості та реалізація застосунку

Висновки

Список використаної літератури та посилань

Додатки

Дата видачі “ \_\_\_\_ ” \_\_\_\_\_ 2020 р. Керівник \_\_\_\_\_  
(підпис)

Завдання отримав \_\_\_\_\_  
(підпис)

**Тема:** Розробка мобільного застосунку для репетиторів

**Календарний план виконання роботи:**

№	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання курсової роботи	28.10.2020	
2.	Пошук літератури за тематикою	15.12.2020	
3.	Ознайомлення з літературою	10.01.2021	
4.	Аналіз існуючих пропозицій	20.01.2021	
5.	Планування структури розділів текстової частини	10.02.2021	
6.	Проектування архітектури практичної частини	21.02.2021	
7.	Створення віддалених місць збереження проекту та налаштування середовища розробки	28.02.2021	
8.	Реалізація серверної частини застосування на базі Node.js	15.03.2021	
9.	Реалізація клієнтської частини застосунку на базі Flutter	05.04.2021	
10.	Написання текстової частини курсової роботи	07.05.2021	
11.	Перегляд змісту виконаної роботи керівником	14.05.2021	
12.	Дооформлення курсової роботи	15.05.2021	
13.	Створення презентації виконаної роботи	16.05.2021	
14.	Здача курсової роботи на перевірку	17.05.2021	

Студент Копійка В.Г.

Керівник Борозенний С.О.

“    ”                      2021 р.

## ЗМІСТ

Анотація .....	6
Перелік Залучених Скорочень .....	7
Вступ.....	8
<i>Актуальність теми та її практична значущість</i> .....	8
<i>Структура роботи</i> .....	9
<i>Постановка задачі</i> .....	10
Розділ 1. Аналіз предметної області.....	11
1.1 Аналіз потреб репетиторів та існуючих рішень-сервісів для них .....	11
1.2 Виклики сьогодення у мобільній розробці.....	13
1.3 Альтернативні підходи до мобільної розробки .....	14
1.4 Висновки до розділу 1 .....	17
Розділ 2. Теоретичні відомості.....	18
2.1 Вибір інструменту для розробки клієнтської частини застосування та відомості про нього .....	18
2.1.1 Огляд популярних фреймворків для розробки мобільного додатку ....	18
2.1.2 Вибір фреймворку Flutter, відомості про нього та його мову .....	20
2.1.3 Особливості фреймворку Flutter та його використання.....	21
2.2 Вибір технології для створення серверної частини застосунку.....	23
2.2.1 Огляд популярних інструментів для розробки серверної частини.....	23
2.2.2 Вибір Node.js та відомості про нього.....	25
2.2.3 Використання Node.js у поєднанні з фреймворком .....	27
2.3.1 Вибір належної БД для застосунку .....	31
2.4 Висновки до розділу 2 .....	32
Розділ 3. Опис реалізації застосування .....	34
3.1 Аналіз технічного завдання .....	34
3.2 Реалізація серверної частини застосування.....	35
3.2.1 Огляд архітектури серверної частини.....	35
3.2.2 Пояснення реалізації багатошаровості серверної частини на прикладі .....	38
3.2.3 Створення авторизації .....	40
3.3 Реалізація клієнтської частини застосування.....	42

3.3.1 Огляд архітектури клієнтського застосування .....	42
3.3.2 Огляд реалізованої архітектури клієнтської частини .....	44
3.3.3 Робота з даними для авторизації дій користувачів .....	46
3.3.4 Реалізація системи нагадувань .....	47
3.4 Тестовий огляд роботи реалізованого застосунку.....	48
3.5 Висновок до розділу 3 .....	53
Список Використаних Джерел.....	55
Додатки.....	61
Додаток А.....	61
Додаток Б .....	62
Додаток В.....	63
Додаток Г .....	64
Додаток Ґ.....	65
Додаток Д.....	66

## Анотація

Мета виконання даної курсової роботи полягає у створенні мобільного додатку, який би зміг стати в нагоді репетиторам та учням під час їхньої взаємодії. Протягом виконання курсової роботи було виконано аналіз предметної області та огляд існуючих сервісів для задоволення потреб репетиторства.

У роботі здійснено вибір інструментів для реалізації застосування та розглянуто їхні особливості. Описана реалізація кожної частини застосунку відповідно до обраних технологій: клієнтська розроблена на базі каркасу Flutter, а серверна – на базі платформи Node.js.

У результаті отримано мобільний додаток, який має функціонал, що здатний полегшити організацію процесу співпраці між викладачами, що надають репетиторські послуги, та їхніми учнями.

Ключові слова: розробка, додаток, застосування, технології, інструменти, репетиторство, мобільні операційні системи, сервіс, розширення.

## Перелік Залучених Скорочень

ОС – операційна система.

UI – від англ. User Interface.

UX – від англ. User Experience.

2D – від англ. 2-Dimensional.

npm – від англ. node package manager.

pub – від англ. publishing.

API - від англ. Application Programming Interface.

JIT – від англ. Just In Time.

AOT – від англ. Ahead Of Time.

FFI – від англ. Foreign Function Interface.

БД – база даних.

СКБД – система керування базами даних.

РБД – реляційна БД.

SQL – від англ. Structured Query Language.

DAO - від англ. Data Access Object.

REST – від англ. Representational State Transfer.

HTTP – від англ. Hyper Text Transfer Protocol.

JSON – від англ. JavaScript Object Notation.

JWT – від англ. JSON Web Token.

CRUD – від англ. Create Read Update Delete.

## Вступ

### *Актуальність теми та її практична значущість*

Створення мобільних додатків стало дуже важливою частиною сучасного світу програмної розробки. Важко уявити користь, що надають нам сучасні пристрої, без застосунків, які на них встановлюються. Навряд чи без них можуть існувати сьогодні галузі, що вимагають безпосередньої взаємодії між людьми.

Поняття дистанційного спілкування та співпраці в цілому набуло нових обертів, коли почався активний розвиток смартфонів та застосувань для них. Тепер для того, аби швидко написати якесь повідомлення або надіслати документ, не треба мати з собою громіздкий комп'ютер. Навіть соціальні мережі, котрі існували ще до появи перших “розумних” телефонів, набули ще більшої популярності в силу зручності того, що почали пропонувати їхні версії у вигляді додатків. Вони почали пропонувати зручну навігацію та приваблювати користувачів новими особливостям, деякі з яких могли б бути доступними навіть без доступу до мережі Інтернет.

Ледь не кожна компанія, котра надавала якісь сервіси у вигляді Інтернет-ресурсів, зрозуміла важливість створення мобільного застосунку. Він би міг надати додаткові можливості, реалізація яких була неможливою в силу обмежень браузерів. До того ж, для багатьох компаній та постачальників сервісів завжди було важливим зробити так, аби їхнім продуктом дійсно регулярно користувалися. Саме можливості пристроїв, особливо наявність у них різного роду датчиків, дозволили повною мірою впоратися із цією задачею.

На жаль, навіть сьогодні існують такі сфери взаємодії між людьми, котрі якщо і мають застосунки, то їх або дуже мало, або вони повноцінно не задовольняють потреби своїх користувачів. Однією з таких галузей є репетиторство, адже переважна більшість “спеціалізованих” сервісів відіграють роль лише Баз знань з інформацією про зареєстрованих викладачів. Вони навряд чи можуть вважатися сервісами, які б дійсно покращили досвід співпраці між учнями та репетиторами.



Така ситуація є досить дивною, адже попит на репетиторів тільки продовжує зростати в силу різних чинників. Насамперед це пов'язано із збільшенням зацікавленості у додатковій позашкільній підготовці до різного роду тестів та оцінювань, які неабияк турбують звичайних школярів, майбутніх абітурієнтів та їхніх батьків. Згідно з інформацією додатку А, що надає український сервіс Serpstat[1] за аналітикою з Google Ads, пошукові запити із ключовими словами “репетитор”, “репетитори” або їм дотичними користуються неабиякою популярністю. Відповідно до особливості показнику частотності появи слова у пошуку за додатком Б, що надається нам у звіті, за місяць здійснюється щонайменше кілька тисяч різних запитів за допомогою Google Search із різних регіонів України на тему репетиторства. Усе це свідчить про немалу актуальність цієї галузі.

З огляду на популярність теми репетиторства і теперішню важливість існування зручного додатку для будь-якої сфери, у якій переважає взаємодія між людьми, метою роботи було вирішено поставити створення мобільного застосунку, який би зробив досвід роботи репетиторів з учнями більш зручним та цікавим.

### *Структура роботи*

Дана робота складається з таких складових: змісту, анотації, вступу, основної частини та висновків до кожного з її розділів, загальних висновків, списку використаних джерел інформації та додатків.

Основна частина складається з трьох розділів.

У першому розділі здійснено аналіз предметної області репетиторства та розглянуто гарний приклад застосунку для цієї області. Також окреслені виклики, із якими стикаються розробники, та визначено деякі підходи до створення мобільних додатків.

У другому розділі виокремлено популярні технології для розробки складових клієнт-серверної архітектури. Також здійснено обґрунтований вибір інструментів для кожної частини застосування.

У третьому розділі проаналізовано технічне завдання та визначено необхідний функціонал застосунку для репетиторів. Також описано повну реалізацію застосування на базі обраних інструментів та підходів.

### *Постановка задачі*

1. Розглянути потреби сфери репетиторства, запропоновані для них сервіси, а також проаналізувати існуючі виклики та підходи у мобільній розробці.
2. Описати вибір технологій та дослідити обрані інструменти для створення додатку.
3. Реалізувати мобільний застосунок для покращення досвіду співпраці репетиторів з учнями за допомогою обраних інструментів.

## Розділ 1. Аналіз предметної області

### 1.1 Аналіз потреб репетиторів та існуючих рішень-сервісів для них

Із самого початку люди зрозуміли наскільки важливо й корисно мати певний сервіс для розміщення та перегляду існуючих репетиторських послуг у межах їхніх міст та областей. Завдяки тому, що мережа Інтернет почала ставати ближчою до людей, поява сайтів, які б стали в пригоді як репетиторам, так і їхнім клієнтам, була лише питанням часу.

Хоч і їхня поява відіграла неабияку роль у полегшенні життя для обох сторін, вони ставали все менш цікавими у користуванні. Згодом такі сайти намагалися набувати додаткових функцій, але переважна кількість з них через це ставала ще важчою у використанні. Відсутність дійсно корисних функцій, які б могли зацікавити саме учнів або їхніх батьків, призводила до того, що про їхнє відвідування швидко забували. Із часом такі ресурси перетворювалися на звичайні електронні довідники із інформацією про зареєстрованих репетиторів. Як наслідок, єдиними функціями, якими користувалися, стали пошук потрібних контактних даних та написання відгуків. Так само й зареєстровані викладачі навряд чи відвідували цей сайт для чогось іншого, окрім як зміни якоїсь інформації про себе або читання відгуків про себе. Хоч згодом і почали з'являтися сайти, які мали перероблений UX, котрий вирішував ледь не найголовнішу проблему заплутаності інтерфейсу попередників, вони все одно не задовольняли потреби користувачів у нових функціях. Причиною слугувала обмеженість того, що могли запропонувати браузері, адже саме це й не давало розробникам створити щось зовсім нове, що могло б змусити аудиторію їхнього продукту зростати.

Пізніше, коли почалася ера невпинного розвитку смартфонів та застосунків для них, тема помічника для репетиторів та учнів набула нового життя. Поява спеціалізованих додатків для організації взаємодії між учнями та викладачами почала вирішувати проблеми, котрі завжди були супутниками усіх сайтів. Найпершим, що одразу вплинуло на ріст популярності таких застосунків

був також користувацький інтерфейс. Компактність смартфонів дозволила раціонально використати вільний простір та уникнути заплутаної навігації, котра раніше неабияк відштовхувала різних користувачів, особливо молодих учнів та людей похилого віку. Другим, але не менш важливим, стала поява нагадувань, котрі підтримувалися зі сторони ОС. Вони дозволяли надсилати користувачеві інформативні повідомлення про прийдешні зустрічі та уроки. Людина навіть могла отримувати ці сповіщення тоді, коли сам застосунок не був запущений на її пристрої. Це стало неабияким помічником як для учнів, так і для викладачів. Для всіх було важливим мати зібрані нагадування в одному місці, не тримаючи в голові можливі зміни в розкладі репетитора або інші додаткові обставини. Тому усі ці функції, що з'явилися внаслідок використання можливостей смартфонів, надали другий шанс таким сервісам.

Чудовим прикладом гарного додатку для репетиторів та учнів є Wyzant[2]. Це досить популярний застосунок у США, кількість завантажень якого налічує більше п'ятдесяти тисяч. Він був розроблений однойменною компанією[3], котра ще в 2005 році почала займатися реалізацією платформи для полегшення взаємодії між тими, хто прагне поділитися знаннями, та тими, хто хоче їх отримати. Дана компанія також почала свій шлях з реалізації сервісу на рівні спеціалізованого сайту, котрий сьогодні налічує неймовірну велику кількість викладачів, а трохи пізніше вона запропонувала й зручний додаток. Wyzant пішла навіть далі, адже вона пропонує не тільки застосунок із унікальними можливостями, котрих так не вистачало раніше у сайтів. Компанія ще забезпечує централізовану перевірку сертифікатів, дипломів та знань всіх тих репетиторів, які в них зареєстровані. Тож їхній додаток задовольняє ледь не все, що тільки може потребувати користувач такої платформи, зокрема наявність зважених рейтингів викладачів, які перевіряються на достовірність, можливість гнучкого планування зустрічей, підтримка нагадувань, організація миттєвої оплати уроку відповідно до вказаної ціни та наявність рекомендацій існуючих пропозицій на базі користувацьких побажань[4].

З огляду на всі згадані особливості, якими володіє цей застосунок, Wyzant є чудовим прикладом того, як сьогодні необхідно робити сервіс для організації навчання між репетиторами та учнями. Цей додаток не тільки користується всіма перевагами, що може надати сучасний мобільний пристрій, але й охоплює велику аудиторію користувачів. Його розробники не оминули увагою жодну з двох найпопулярніших ОС смартфонів[5], адже цей додаток існує як на iOS, так і на Android, пропонуючи максимально ідентичний функціонал для обох систем. До того ж, рейтинг Wyzant на цифрових майданчиках цих двох ОС говорить сам за себе, маючи оцінку більше чотирьох з п'яти можливих і там, і там. Це підтверджує як високу якість розробленого продукту, так і зразкову актуальність того, що він та йому подібні додатки мають пропонувати.

## 1.2 Виклики сьогодення у мобільній розробці

У розробників мобільних застосунків із кожним роком з'являється все більше викликів, із котрими вони мають стикатися, аби запропонувати дійсно гарний продукт. Слід зауважити, що кожного дня на цифрових майданчиках розповсюдження додатків з'являється багато нових застосунків, зокрема на App Store від Apple із кожним днем таких стає на сотні більше[6]. Всі вони мають на меті привернути увагу якомога більшого кола користувачів, але лише обрані дійсно чогось варті й стають популярними. Можна сказати, що головну роль в успіху додатка насамперед відіграє ідея або його орієнтована галузь, але навіть це з поганою реалізацією навряд чи стане чимось більшим за черговий застосунок. Саме тому розробникам неабияк необхідно подумати над тим, аби їхній додаток був достатньо раціональним у реалізації за більшістю критеріїв, перш ніж приступати до його створення, адже всі розуміють ризики та необхідність у майбутній підтримці. Як наслідок, сьогодні у програмістів з'явився вибір кількох підходів, за допомогою яких вони можуть почати втілення в життя власного продукту.

Найпершим таким запропонованим варіантом у черзі є авжеж нативна (від англ. native - рідний) розробка під конкретну ОС. Вона передбачає, що застосунок створюється за допомогою певного набору ПЗ, що надається розробниками цієї системи. Слід виокремити, що така розробка дуже вітається та підтримується зі сторони компаній, які і є власниками тих чи інших платформ. Вони зацікавлені в тому, щоб весь процес створення застосунку перебігав у їхньому середовищі, яке вони з легкістю можуть підтримувати. Як наслідок, це призводить до того, що застосунки, які написані таким чином, не тільки використовують всі наявні можливості пристрою і мають гарну працездатність, а й легше проходять публікацію на рівні магазину додатків[7].

На жаль, такий підхід не завжди підходить деяким компаніям та командам програмістів. Усі розуміють важливість забезпечення того, аби якомога більше користувачів змогли принаймні завантажити додаток на свій телефон, а для цього потрібно надати застосунок хоча б на головних мобільних ОС сучасного ринку, зокрема на таких найпопулярніших, як Android та iOS[5]. На жаль, створення окремих нативних програм для двох ОС може неабияк вплинути на виділений бюджет. Це може призвести до уповільнення розробки через різні фактори, зокрема внаслідок пошуку додаткових фахівців, які б розумілися на обох системах, навчання програмістів або відповідне розширення штату працівників під кожен з ОС[8]. Саме через ці аспекти досі ведуться дискусії з приводу оптимальності того чи іншого підходу[9]. Сьогодні навряд чи хтось візьме на себе відповідальність сказати те, що нативна розробка є найкращим і єдиним правильним вибором для всіх.

### 1.3 Альтернативні підходи до мобільної розробки

На противагу нативному створенню застосунків з'явилася інструменти, що зробили можливою кросплатформну (від англ. cross-platform – міжплатформний) розробку, котра часто передбачає написання коду переважно однією мовою. Сутність такого підходу полягає насамперед у створенні одного додатку, котрий

зміг би існувати та запускатися одночасно на кількох ОС, але при цьому розробники матимуть підтримувати лише “один код на всіх”.

Цікаво, що внаслідок використання цього підходу з’явилося кілька типів міжплатформних додатків, котрі мають певні відмінності. Серед них можна виокремити такі, як PWA, гібридні, інтерпретовані та компільовані міжплатформні додатки.

Не так давно з’явилися застосування під аббревіатурою PWA. Хоч процес їхньої розробки нагадує веб-сайти, вони можуть встановлюватися на всі доступні ОС у вигляді додатків, але в обхід цифрових магазинів. Тож система сприймає їх як звичайні застосунки, що можуть використовувати деякі її додаткові можливості, зокрема отримувати сповіщення в реальному часі та працювати без доступу до мережі[10]. Це все можливо завдяки залученню скрипту (від англ. script – сценарій) Service Worker, котрий у свою чергу дозволяє обробляти та перехоплювати мережеві запити у фоновому режимі, адже саме він дозволяє виконуватися коду у фоні, а також керувати кешуванням ресурсів. Він дозволяє користуватися вже існуючими даними, котрі оновлюватимуться, як тільки буде можливість. Це дозволяє відтворювати особливості звичайного додатку, що не вимагає для свого запуску та базових потреб доступу до мережі[11]. Отже, враховуючи певні ознаки, PWA посідають місце між сайтами та “реальними” застосунками.

Гібридні застосунки також неабияк базуються на тому, як влаштовані сайти. Головна відмінність криється у тому, що, на відміну від звичайного використання браузера або ж завантаження чогось на кшталт додатку PWA, вони традиційно завантажуються з цифрового магазину застосунків тієї чи іншої ОС[12]. Насправді ж “гібриди” мають у собі прихований веб-застосунок, що вимальовується усередині спеціального контейнеру (від англ. container – вмістилище), що передбачений ОС. Такий блок виступає в ролі вбудованого браузерного компонента, зокрема як WebView[13] на Android. Саме він і дозволяє відображати веб-сторінки, приховуючи окремі елементи інтерфейсу звичайних браузерів. Серед переваг над звичайними веб-застосунками можна

виділити те саме, чим можуть зацікавити розробників і PWA-додатки. Цей застосунок не вимагатиме доступу до мережі, аби запуститися. Більше того, така розробка вже відкриває можливості використання додаткових частин системи пристрою, зокрема доступ до даних контактів, нотаток, фото і т.д., що збережені на пристрої. Це стало можливим завдяки API, що беруть на себе роль FFI[14], які надаються інструментами для гібридної розробки. FFI виступатиме у ролі такого собі “містка” між компонентом WebView та нативним кодом системи. Він дозволяє викликати функції та надавати в користування структури, що належать рідній мові системи. Єдиним можливим недоліком такого підходу є важкість створення відповідного системі UI, котрий би не суперечив наставленням розробників ОС. Адже під час створення гібридного застосунку розробник має лише можливості, що надані інструментами для розробки веб-інтерфейсів[15].

Інтерпретовані додатки сьогодні користуються неабиякою популярністю серед розробників. Їхня особливість полягає в тому, що такі застосунки здатні вимальовувати компоненти рідного інтерфейсу ОС. Вони вже не спираються на спеціальні блоки для відображення інтерфейсу веб-сторінок, але для того, аби забезпечувати обмін даними з мовою системи, вони також використовують FFI. Робота таких фреймворків (від англ. framework – каркас) для розробки інтерпретованих програм полягає в тому, що завдяки системним інтерпретатором мови на кшталт JavaScript вони можуть отримувати перетворені з мови фреймворку компоненти інтерфейсу відповідні ОС. Відбувається конвертація таких компонентів у ті, що підходять певній системі[15].

Також багато уваги привертають на себе каркаси, що полягають у створенні компільованих застосунків. Такі додатки вже не покладаються ні на WebView, ні на вбудовані системні інтерпретатори інших мов для відображення користувацького інтерфейсу. Мови таких фреймворків компілюються в системний байт-код, що потім виконується без втрати часу на FFI та інтерпретацію[15].

Тож ми переконалися, що існує достатня кількість різноманітних підходів до створення міжплатформних застосунків, тому кожний розробник може



підібрати оптимальних підхід відповідно до власних потреб, умов, можливостей та вимог роботодавця. У часи, коли на вагу золота кожна хвилина, а зайві витрати капітальних ресурсів є фатальними, гарні міжплатформні рішення є ледь не панацеєю для невеликих компаній та стартапів.

#### 1.4 Висновки до розділу 1

У цьому розділі були розглянуті передумови появи сервісів для покращення взаємодії між репетиторами та учнями, а також їхні недоліки та проблеми, що впливали на UX. Був здійснений аналіз застосунку, що виправив більшість проблем попередників, на прикладі Wyzant, який зараз користується популярністю.

Були окреслені проблеми, що сьогодні виникають у творців мобільних додатків, та дилему вибору оптимального підходу до створення застосувань. Крім того, було розглянуто кілька існуючих типів застосунків.

## Розділ 2. Теоретичні відомості

### 2.1 Вибір інструменту для розробки клієнтської частини застосування та відомості про нього

#### 2.1.1 Огляд популярних фреймворків для розробки мобільного додатку

Сьогодні розробники як ніколи розуміють те, наскільки важливо створити такий продукт, котрий би одночасно задовольняв їхніх клієнтів і був вигідним у сенсі можливих витрат на розробку та підтримку. Водночас розробникам дуже важливо забезпечити існування застосунку щонайменше на кількох ОС. Така особливість безперечно дозволить йому стати більш поширеним серед набагато ширшого кола користувачів. Утім, як ми вже переконалися, перед ентузіастами, що прагнуть створити власний мобільний застосунок, постає питання раціональності окремої розробки під кожен ОС.

За останні роки кілька технологічних гігантів почали демонструвати велику зацікавленість у тому, аби запропонувати інструмент, котрий би “затмарив” окрему розробку. Насамперед відомо, що такі компанії, як Google, Facebook, Microsoft та Adobe, намагаються запропонувати своє бачення такої пропозиції. Як не дивно, але до того, що вони пропонують здебільшого належать інструменти, які і користуються популярністю серед розробників. Причина такої довіри полягає насамперед у “впевненості у наступному дні”, адже повноцінна розробка та постійна підтримка від якоїсь великої компанії неабияк впливає на вибір інструменту. І тут справа не лише у надійній опорі, що має той чи інший інструмент, а й у відповідних перевагах, котрі не завжди притаманні пропозиціям від звичайних ентузіастів. До таких плюсів передусім належать регулярні виправлення помилок та наявність змістовної документації, а деякі каркаси навіть пропонують щось на кшталт власних відео-курсів на хостингу YouTube. Тож для багатьох розробників буде логічним зробити вибір на користь інструменту з подібним підґрунтям.

Існування чималої кількості пропозицій на ринку інструментів для міжплатформної розробки мобільних застосунків дозволяє визначити ті, які

набули або ж ще тільки набувають популярності серед розробників. Для усвідомлення картини уподобань та вимог програмістів щодо поняття “гарного” каркасу для міжплатформної розробки, нам слід розглянути найпопулярніші інструменти. За інформацією статистичних даних від компанії JetBrains, отриманих внаслідок опитування The State of Developer Ecosystem 2020[16], найвищі позиції займають у порядку послідовності React Native від Facebook, Flutter від Google та Cordova від Adobe. Цікаво, але дана трійка перших місць не змінилася з моменту такого ж самого опитування 2019 року[17]. Оскільки результати переможців уже протягом двох років не змінюються, то можна зробити висновок, що дана інформація відповідатиме досить репрезентативним реаліям останніх часів, а не тимчасовою популярністю того чи іншого фреймворку. Тож можна розглянути трійку каркасів-переможців у ролі головних інструментів на ринку кросплатформної мобільної розробки. Відповідно до них можна також сказати й про те, які саме підходи до створення застосунків є більш поширеним серед програмістів. Додатки на React Native належать до інтерпретованих, на Flutter - до компільованих, а на Cordova - до гібридних[15].

Одним з найостанніших та відповідно наймолодших інструментів, який набирає неабияких обертів, є саме Flutter, але при цьому він вже жодним чином не поступається іншим у своїй популярності та залученості спільноти. Статистика відкритого репозиторію на GitHub цього фреймворку[18] свідчить про неабияку зацікавленість у ньому, зокрема на момент січня 2021 року маємо такі показники:

- кількість користувачів, що зробили внесок в розробку ~ 780
- кількість релізів версій ~ 400
- кількість спостерігачів за каркасом ~ 3 400
- кількість вподобань ~ 111 000

Це дуже гарні результати, а кількість людей, що вподобали цей інструмент, вже обійшла відповідні показники того ж самого періоду часу, що мали репозиторії каркасів React Native[18] та Cordova[19].

### 2.1.2 Вибір фреймворку Flutter, відомості про нього та його мову

Flutter передбачає написання коду мовою Dart, що також розроблена Google. Саме через неї фреймворк завдячує багатьом своїм особливостям, котрі привертають увагу ентузіастів, зокрема можливість оновлення “на-льоту” та компіляція мови в машинний код є одними з них[20]. Ледь не найголовнішим аспектом, на думку самої Google, у каркасу є саме те, що всю відповідальність за відмалювання компонентів інтерфейсу бере на себе окремий двигун для рендерингу 2D графіки. Саме це й дозволяє досягти покращення працездатності та максимально наблизитися до плавності інтерфейсу нативних застосунків. Також слід згадати й про наявність власного пакетного менеджера під назвою pub[21], котрий постійно розширюється новими бібліотеками та віджетами як від спільноти, так і від самої команди творців. Більше того, все, що є в цьому менеджері, час від часу проходить перевірку на якість коду та документації, внаслідок чого все має власну оцінку якості від працівників Google, а це також неабияк імponує програмістам.

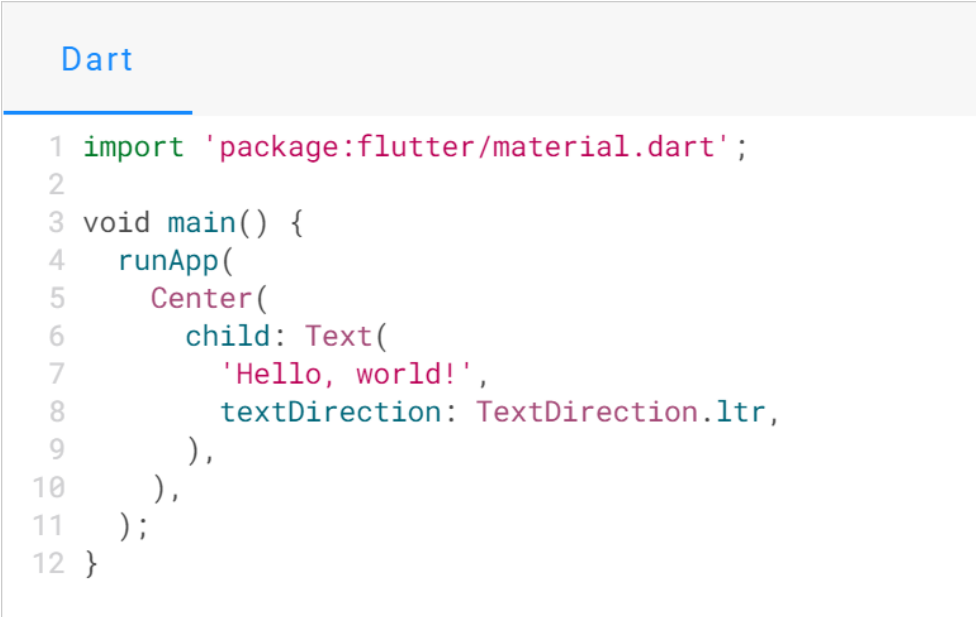
Цікаво, що від самого початку свого анонсу розробники, які цікавляться створенням мобільних програм під різні ОС, плекають неабиякі на дії на цей інструмент, вважаючи його гарним конкурентом інших вже досить відомих аналогів. Сама ж Google також обіцяє, що Flutter стане головним інструментом для розробки інтерфейсу та додатків для їхньої майбутньої ОС під кодовою назвою Fuchsia[22], а це теж підігріває зацікавленість публіки у ньому. Про неабиякий ріст популярності свідчить і кількість написаних на Flutter застосунків, які поступово з’являються. Із-поміж тих, що мають велику аудиторію слід згадати такі: інтерактивний журнал-нотатник Reflectly, один із найпопулярніших агрегаторів знижок та цін у США - Groupon, багатьом відомий майданчик Ebay та кілька власних застосунків Google, серед яких Google Assistant та Google Ads[23].

Аналізуючи багатообіцяюче підґрунтя програмного каркасу Flutter, структурованість його документації та зростаючу спільноту, він стане ледь не найкращим вибором для знайомства з мобільною розробкою.

Dart - це невід’ємна частина каркасу Flutter. Це однопоточна та статично типізована мова, що базується на підтримці асинхронного програмування та має C-подібний синтаксис. Вона також підтримує використання динамічних типів, а це, за потреби, може стати деяким розробникам до смаку. Більше того, Dart має різні типи компіляторів, що можуть також стати в нагоді. Наприклад, для тих, хто спиратиметься на створення застосунків для запуску на пристроях, він пропонує одразу два типи компіляції для задоволення додаткових потреб розробників. Зокрема є можливість використання JIT компіляції, що дозволяє реалізувати у Flutter оновлення “на-льоту” без перебудови проекту під час виконання шляхом компіляції в машинний код лише тих частин, що були змінені. Другим же обов’язковим типом є AOT, що компілює весь проект в машинний код під час його повної побудови, передуючи виконанню. Саме таким чином відбувається компіляція застосунка в архів-виконувач, який потім може встановлюватися на пристроях користувачів. Також Dart має компілятори, що дозволяють перетворювати код цієї мови у відповідний JavaScript[24].

### 2.1.3 Особливості фреймворку Flutter та його використання

Основною частиною роботи з інструментом Flutter є головним чином використання його віджетів (від англ. widget – пристосунок). Весь принцип створення інтерфейсу користувача базується на побудові дерева з таких елементів. Будь-яке дерево має обов’язково починатися з визначеного кореневого віджета, котрий має приймати відповідна функція “runApp”, що обов’язково викликається саме у точці входу програми, як на рисунку 1.



```

Dart
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(
5     Center(
6       child: Text(
7         'Hello, world!',
8         textDirection: TextDirection.ltr,
9       ),
10    ),
11  );
12 }

```

Рисунок 1 - Приклад простої програми “Hello World” [25]

Кожний віджет такого дерева може відігравати роль компоненту, стан якого може динамічно змінюватися, трансформуючи власний опис. Саме зміна цього опису дозволяє фреймворку визначати головні нововведення, що були здійснені в порівнянні з попереднім станом дерева. Це дозволяє з легкістю визначати тільки необхідні місця для перетворення, уникаючи зайвого перемальовування всього інтерфейсу.

Flutter одразу “пропонує” велику бібліотеку вже готових віджетів, не враховуючи вміст наробок спільноти в його пакетному менеджері pub. Кожний із пристосунків має певні властивості, котрими можна визначати та змінювати його вигляд. Існує велика кількість таких елементів, що неабияк пришвидшують розробку інтерфейсу без зайвого вигадування. Розробники також можуть створювати і власні віджети, комбінуючи вже існуючі та змінюючи їхні особливості. Зокрема програміст має можливість створити новий віджет або ж шляхом наслідування безстанового (від англ. *stateless* – відсутність стану) віджету під назвою *StatelessWidget*, або ж станового (від англ. *stateful* – наявність стану) *StatefulWidget*, що підтримуватиме динамічну зміну власного опису під час свого існування. Якщо ж розробник зацікавлений у тому, аби встановлювати та оновлювати вигляд якогось елементу UI внаслідок взаємодії або з боку

користувача, або ж з боку самої програми, то йому на допомогу прийде наслідування саме другого варіанту. Це дозволить впливати на вигляд такого компонента протягом всього його життєвого циклу, зокрема від початкового етапу створення до наступного перемальовування або ж остаточного знищення.

Слід також згадати й про те, що існує досить велика кількість зручних компонентів для організації розташування елементів на екрані, що дозволяють робити різні комбінації відображення елементів на ньому. До них належать такі, як `SafeArea`, що дозволяє уникнути перекривання віджетами статус-бару та навігаційної стрічки пристрою, `Stack`, котрий відкриває можливості довільного позиціонування будь-якого компоненту на доступній площині, `Row` та `Column`, що також заощаджують час розробнику, автоматично розташовуючи у визначеному порядку обрані елементи. І це ще далеко не всі корисні віджети, які може запропонувати цей інструмент. Більше того, Flutter одразу дозволяє використати певну бібліотеку відповідно до платформи, для якої відбувається розробка. Таким чином, існують окремі пакети компонентів із дизайном, що відповідає філософіям платформи Android та iOS. Усе, що потрібно для цього зробити, це просто імпортувати необхідний пакет, після чого елементи автоматично отримають потрібний вигляд.

Отже, сам інструмент може бути досить гнучким під час розробки на ньому UI мобільного застосунку, адже він не тільки пропонує багато готових рішень, а ще й дозволяє програмісту обрати власний шлях побудови інтерфейсу для додатку. До того ж існує достатньо велика база документації від Google, що неабияк дозволить поринути в те, що дозволяє створювати Flutter.

## 2.2 Вибір технології для створення серверної частини застосунку

### 2.2.1 Огляд популярних інструментів для розробки серверної частини

Сьогодні розробникам як ніколи варто зробити правильний вибір під час обрання технологій для створення серверної частини своїх застосувань. Адже саме від неї залежатиме якість і загальна працездатність сервісів, що вони

надаватимуть. Зараз важко уявити якийсь успішний додаток, який використовує лише локальні можливості девайсу (від англ. device – пристрій) і не має жодної взаємодії із якимось віддаленим пристроєм чи то для отримання інформації, чи то для її збереження. Хоч зараз і вибір існуючих інструментів для реалізації бекенду (від англ. backend) досить широкий, перед розробниками як завжди постає проблема доречного вибору як з урахуванням потреб їхнього майбутнього продукту, так і з повноцінним задоволенням власних розробницьких побажань та можливостей.

Серед найпопулярніших фреймворків особливо останнього часу дуже сильно відзначають ті, що базуються на Node.js, Python, Ruby, PHP та Java, зокрема до них входять Express, Django, Rails, Laravel та Spring відповідно[26]. Усі вони вже давно завоювали власні аудиторії, котрі за ним слідкують і часто використовують у своїх проектах.

Зокрема фреймворк Express відомий своєю легкістю у використанні та достатньою документованістю. Він не перестане набирати обертів доти, доки користуватиметься прихильністю Node.js, що неабияк вплинув на збільшення аудиторії розробників бекенду. Сьогодні Node.js у поєднанні з різними каркасами зайняв своє місце у розробках таких компаній, як PayPal, Uber та LinkedIn[26].

Так само й фреймворк Spring відмовий своєю потужністю у вигляді бібліотек та шаблонів, що дозволяють досягти потрібних результатів. Він давно став ледь не найдоречнішим інструментом для застосування у дійсно великих проектах, налічуючи такі як Netflix[27]. Крім того, команда розробників цього фреймворку неабияк пишається рівнем якості інструменту, відзначаючи серйозний акцент на безпеці й перевірності всіх бібліотек[28].

Django, у свою чергу, також став улюбленцем через наявність додаткових функцій захисту, зокрема готові засоби запобігання SQL-ін'єкціям та XSS-атакам, що надаються, є важливими заощадниками часу для багатьох програмістів[29].



Так само Rails та Laravel мають досить великі кола прихильників. Обидва відомі своєю “дружністю” до новачків та досить усталеними спільнотами, що мають відкриті форуми, на яких обговорюють все, що тільки може прийти на думку під час розробки бекенду на їхній основі.

Тож, як вже й зазначалося, на вибір у програмістів є достатня кількість дійсно якісних і вагомих технологій для розробки серверної частини. Кожний із інструментів може запропонувати щось своє, незважаючи на можливі недоліки. Залишається обрати лише найоптимальніший.

### 2.2.2 Вибір Node.js та відомості про нього

Напевно для багатьох програмістів найважливішими під час обрання бекенд-технології критеріями, крім її можливостей, є легкість опанування інструменту, залученість спільноти в його розвиток та змістовність офіційної документації. Як наслідок, найоптимальнішим варіантом вибору для розробки серверної частини клієнт-серверної архітектури застосунку буде саме Node.js у поєднанні з таким фреймворком, як Express.

Node.js - це асинхронне JavaScript-середовище, що базується на подієво-орієнтованій архітектурі (від англ. event-driven – керований подіями), для побудови серверних застосувань[30]. Саме ці аспекти, особливо залучення такої асинхронної мови, як JavaScript, роблять використання цієї технології в побудові серверу дещо особливим та відмінним від усталених підходів інших інструментів. Головною перевагою проектів, що розроблені на Node.js, є можливість охоплення великого обсягу одночасних підключень до серверів без відчутних втрат працездатності та швидкості. Відомо, що більшість конкуруючих інструментів базується на тому, що кожне нове підключення має супроводжуватися створенням окремого потоку, що неабияк впливає на використання ресурсів. У той самий час мережеві застосунки на Node.js в силу мови, яку він використовує, працюють протилежним чином. Залучення неблокуючих (від англ. non-blocking – неблокований) викликів до функцій

введення та виведення дозволяє серверам із легкістю впоратися із великою кількістю паралельних запитів до них. Адже не відбувається очікування завершення обробки запиту перед переходом до наступного (див. рисунок 2). Тож середовище “зверне увагу” на запит, коли той у потрібний момент сповістить про подію закінчення його опрацювання. Таким чином, Node.js уникає витрат на очікування відповіді та перемикання між різними потоками під час опрацювання кожного зі звернень.

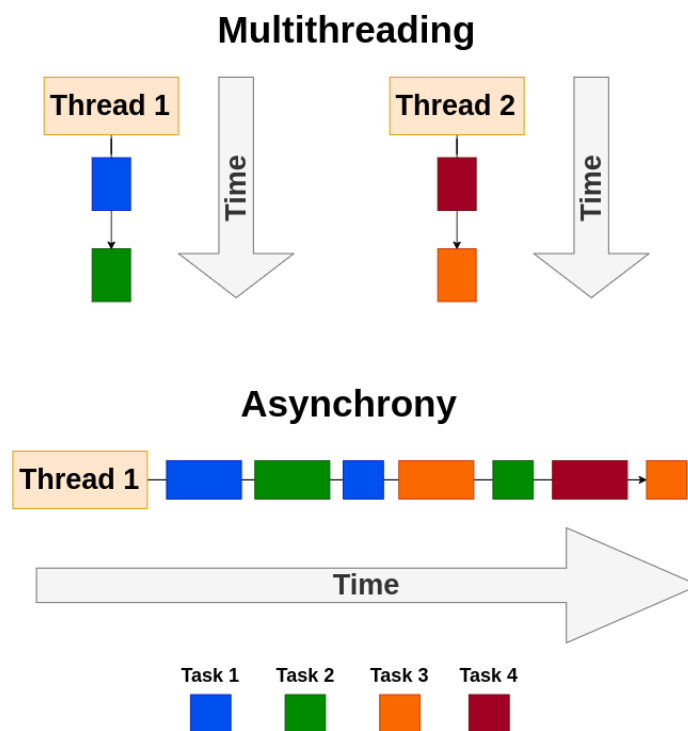


Рисунок 2 - Багатопоточний та асинхронний підхід [31]

Інколи така особливість використання лише одного потоку може мати супутні недоліки. Зокрема слід уникати обрахунків, що можуть залучати важкі для системи операції, та слідкувати за викиданням помилок, адже кожна така ситуація супроводжуватиметься блокуванням потоку[32].

Як наслідок, зважаючи як на згадані переваги, так і на недоліки, це середовище здобуло своїх прихильників у галузях, що базуються на отриманні та обміні невеликими обсягами інформації великою кількістю користувачів. До них здебільшого належать соціальні мережі на кшталт LinkedIn, котра, як уже було зазначено, і використовує Node.js. Саме тому його використання в серверній частині застосунку, що призначений для співпраці між репетиторами

та учнями, буде досить доречним, адже функціонал та потреби точно будуть перетинатися в певних межах з тим, що має пропонувати соціальна мережа.

### 2.2.3 Використання Node.js у поєднанні з фреймворком

За бажання, за допомогою “чистого” Node.js можна реалізувати будь-які можливості на стороні серверу, але це авжеж вимагатиме додаткових зусиль. Для цього на допомогу приходить власний пакетний менеджер під назвою npm[33], що налічує в собі тисячі різних бібліотек, каркасів та шаблонів, що дозволять пришвидшити поринання в розробку як новачкам, так і вже досить досвідченим програмістам. Більше того, цей пакетний менеджер має зручний сайт, на котрому можна дізнатися більше про будь-яке наявне у ньому розширення із відповідними посиланнями на репозиторій в GitHub або GitLab.

Одним із найперших таких каркасів, без котрих важко уявити сучасну розробку за допомогою середовища Node.js, є Express. Сам Express позиціонується в ролі мінімалістичного інструменту, що передусім дозволяє з легкістю писати потрібні обробники для запитів відповідно до кожного методу мережевого протоколу передачі даних HTTP. Також він дозволяє гнучко створювати відповіді на базі шаблонізаторів для створення динамічних сторінок. Багато ентузіастів відчули гнучкість, котру пропонував їм цей інструмент у вигляді підтримки проміжних обробників, що безпосередньо мають доступ до всіх отриманих даних від запиту та об’єкту відповіді на нього. Згодом програмісти почали створювати власні розширення для інструменту, котрі почали задовольняти ще більше потреб під час написання серверної частини. Зокрема з часом з’явилися десятки різних пакетів, що спрощували роботу з отриманням параметрів з URL, перевіркою отриманих даних на можливу небезпечність, встановлення авторизації дій користувачів зі сторони серверу і т.д.[34]. Усі ці аспекти зробили його одним із найпопулярніших інструментів, що існують у пакетному менеджері Node.js, кількість користувачів якого за весь час

його існування, за статистикою самого npm, налічує більше шістнадцяти мільйонів[35].

Використання Node.js разом із Express дозволяє “підняти” найпростіший сервер за лічені хвилини. Для того, аби почати користуватися інструментом, спочатку необхідно ініціалізувати середовище для роботи з менеджером npm шляхом написання команди “npm init” у консольному рядку вашого проекту. Пізніше у вхідному файлі вашого серверного застосунку треба імпортувати необхідну бібліотеку з назвою фреймворка, який розробник має попередньо завантажити та встановити за допомогою згаданого раніше пакетного менеджера. Наприклад, для завантаження Express необхідно знову ж таки в командному рядку виконати “npm install express”. Після того, як потрібний фреймворк вже встановлено, розробник може приступати до реалізації простого серверу. Для цього потрібно викликати на екземплярі express функцію “listen” із вказанням відповідного порту, який “слухатиме” сервер і на якому він запускатиметься. Після цього ж можна перейти до створення маршрутизації, вказуючи назву функції у вигляді відповідного методу HTTP разом із адресою, за якою цей запит має здійснюватися. Вже у середині цієї функції розробник має право визначати відповідь, яка буде надсилатися після обробки цього запиту (див. рисунок 3).

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('hello world')
})

app.listen(3000)
```

Рисунок 3 - Приклад найпростішого серверу на базі Node.js з Express [36]

Як наслідок, створити сервер дуже просто, а ось його масштаб та складність обумовлюється лише тим, чого прагне досягти сам розробник. Уже на

ньому лежатиме реалізація архітектури, залучення додаткових розширень і реалізація власних проміжних обробників.

### 2.3 Загальний огляд типів БД для використання в проекті

Коли настає час обирати БД для застосування в проекті, то розумієш, що навряд чи є погані варіанти, адже скоріше існують лише ті, що підходять в реалізації тих чи інших потреб, та ті, які просто не підпадають під потрібне спрямування. До головних типів, за котрими поділяють бази даних, відносяться реляційні (від англ. relational - відносний) та відповідно нереляційні (від англ. non-relational – невідносний) БД.

Особливість перших полягає в тому, що вони базуються на реалізації реляційної моделі у відповідності до предметної області. Ця модель в свою чергу складається із взаємопов'язаних між собою реляційних відношень (таблиць). Кожне з них має усталену схему з вказаними іменами доменів, допустимі значення яких повинні належати встановленому типу даних. У той час як головною відмінною властивістю других є те, що вони не покладаються на поняття реляційної моделі і не зберігають інформацію у таблицях зі строго визначеною схемою. Структури їхніх даних не залежать від визначених схем або набору допустимих полів та значень, які ці поля можуть набувати[37]. Наприклад, деякі з нереляційних БД спираються на використання документів замість таблиць. Вони дозволяють зберігати в собі найрізноманітніші дані у вигляді списків або ж навіть окремих об'єктів із власними полями, а таке в таблицях реляційних баз даних існувати не може (див. рисунок 4). Також існують і такі, що зберігають інформацію у взаємопов'язаних нодах (від англ. node - вузол) також із можливістю збереження даних довільних типів.

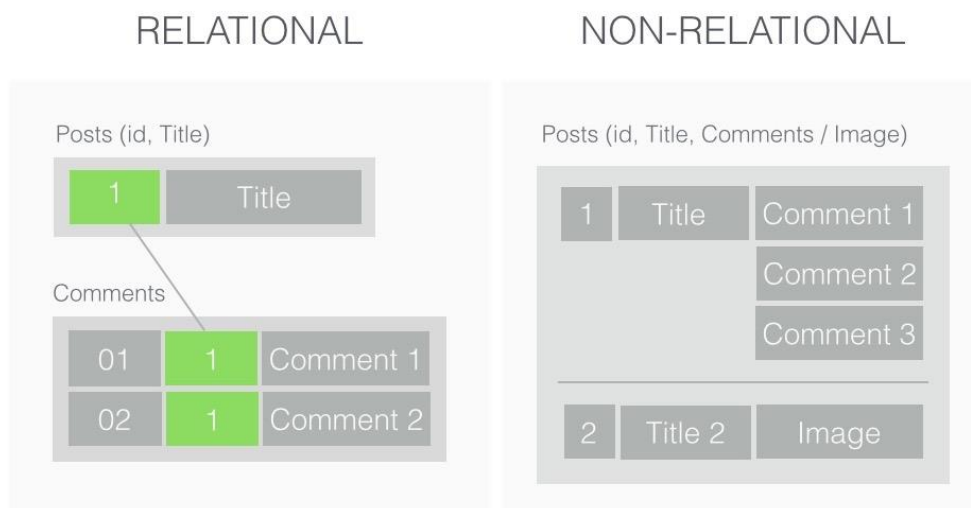


Рисунок 4 - Приклад збереження даних в різних БД [38]

Тож обидва типи БД мають досить кардинальні відмінності в тому, як вони працюють з даними, що й дозволило розділити їхнє призначення. Таким чином, реляційні бази даних набули широкого використання в тих сферах, де цілісність даних та їхня структурованість відіграють переважну роль. У той час як нереляційні аналоги підійдуть більше тим, хто матиме справу зі збереженням навпаки неструктурованих даних, що постійно змінюватимуться. До того ж, вони користуються популярністю у тих випадках, коли треба швидко дістати суттєвий обсяг інформації. Це можливо знову ж таки завдяки особливості збереження в них різного роду даних. У деяких випадках це вагомий аспект, котрий переважає над тим, що притаманно реляційним БД. Адже ті витрачають багато ресурсів на об'єднання великої кількості таблиць для пошуку потрібних рядків. Отже, вибір бази даних залежить насамперед від того, яким чином розробникам потрібно буде організовувати роботу з інформацією у конкретному проєкті. До того ж, програмісти дуже часто переходять до одночасного використання кількох БД для ефективної реалізації різного функціоналу в застосуванні, чого неможливо досягти використанням лише однієї.

### 2.3.1 Вибір належної БД для застосунку

Зважаючи на те, що необхідно реалізувати застосунок із функціоналом, який би відповідав базовим вимогами соціальної мережі для взаємодії між репетиторами та учнями, слід проаналізувати, які інструменти можуть доречно підпасти під потреби.

Такий застосунок повинен зберігати інформацію профілів відповідно до наданих користувачами даних про себе. Оскільки набір відповідних атрибутів між користувачами навряд чи повинен різнитися, буде доречним для збереження інформації використовувати саме реляційну БД. До того ж, враховуючи можливу реалізацію чогось подібного до нескладних дописів із боку репетиторів, наприклад, для вказання відомостей про пропоновані дисципліни, буде корисно звернутися до реляційних таблиць для зручнішого структурування. По-перше, схема відношення такого допису навряд чи колись зазнаватиме змін, а, по-друге, у такому разі кожний із них буде мати можливість тримати в собі посилання на ID свого створювача у вигляді зовнішнього ключа. До того ж, в перспективі можна скористатися стратегіям забезпечення цілісності посилань, які можуть неабияк допомогти під час майбутнього масштабування застосунку. Саме тому для організації та підтримки РБД, що забезпечуватиме збереження такої нескладної структурованої інформації, підійде СКБД, наприклад, MySQL.

MySQL - це досить потужна система керування базами даних від Oracle, що є однією з найрозповсюдженіших у використанні на ринку. За результатами опитування, що здійснила команда відомого форуму StackOverflow, останні два роки переважну кількість прихильників має саме ця СКБД відповідно до відповідей більше ніж сорока дев'яти тисяч респондентів[39]. Це свідчить про наявність неабиякої підтримки зі сторони спільноти, незважаючи на існування великої кількості “реляційних” та інших конкурентів. Серед переваг, що так привернули увагу розробників, насамперед є доступність на всіх платформах, перевіреність на вразливості “витоку даних”, підтримка багатопоточності[40] та

авжеж умовна безкоштовність. Ці та інші особливості роблять MySQL поширеною як у комерційному, так і навчальному використанні.

Також варто розглянути залучення й іншої нереляційної БД для ефективної реалізації деяких особливостей застосунку. Наприклад, якщо розробник має на меті втілити в життя “рекомендаційний двигун” на базі якихось якостей користувачів або інших речей, аби застосування робило доречні пропозиції в реальному часі, йому варто звернутися до графової бази даних, наприклад, Neo4j. Особливість такої БД полягає в тому, що всі дані зберігаються у вузлах, що пов’язані між собою містками у вигляді ребер. Кожний її вузол має в собі безпосередні вказівники на всіх інших, із котрими він був пов’язаний[41]. Саме тому використання цієї технології дуже сильно спрощує пошук складних ланцюжків між даними, що збережені у вузлах. Те, як влаштована Neo4j, дало їй змогу посісти особливе місце в сучасній організації як для отримання релевантних даних для рекомендацій, так і просто для збирання найостанніших результатів взаємодії користувачів з різними частинами сучасних сервісів. Зокрема рекомендації нових друзів у соціальних мережах, кінострічок, що можуть припасти до смаку, та пропозиції пошуку за користувацькою історією набули нових обертів завдяки їй. Отже, графова БД дозволяє робити досить складні запити пошуку даних за певними критеріями без зайвих витрат, що в силу своїх особливостей реляційні та інші нереляційні аналоги, роблять дещо повільніше. Саме тому, якщо в застосунку будуть існувати зв’язки між великою кількістю користувачів або розробники зацікавлені в можливій перспективі впровадження додаткових функцій у вигляді рекомендацій, то залучення для таких потреб ще однієї БД у вигляді Neo4j є досить доречним.

## 2.4 Висновки до розділу 2

Тож у цьому розділі ми розглянули існуючі інструменти, які користуються прихильністю серед великого кола розробників і які можуть гарно підійти для реалізації всієї архітектури нашого проекту. Здійснено відповідний вибір таких



технологій, котрі б задовольнили потребам предметної області застосунку для організації репетиторства, та розглянуто їхні особливості. Як наслідок, маємо повний набір інструментів, за допомогою якого можна перейти до безпосередньої реалізації мобільного додатку відповідно до трирівневої клієнт-серверної моделі, що передбачає використання таких окремих рівнів: застосунку клієнта, серверного застосування та серверу бази даних[42]. Це дозволить досягти якісного розподілу відповідальності між різними частинами нашої архітектури, уникаючи перевантажень для жодної з них та полегшуючи розробку застосування в цілому.

Таким чином, клієнтську частину мобільного застосунку буде реалізовано на мові Dart у поєднанні з фреймворком Flutter для кросплатформної розробки додатків. Серверну ж частину, яка буде відповідати за роботу з БД, отримання, обробку та надсилання даних клієнту, буде створено на базі середовища Node.js та його фреймворку Express. За збереження необхідних даних та їхніх описів відповідатимуть системи керування базами даних MySQL та Neo4j відповідно (див. рисунок 5). Отже, маємо всі інструменти для переходу до наступного розділу курсової роботи.

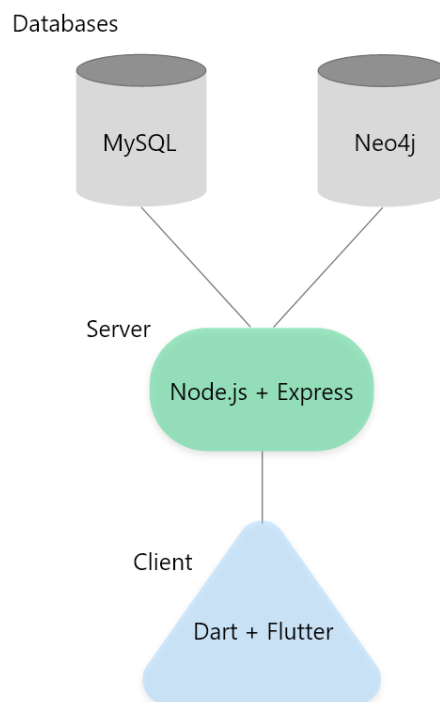


Рисунок 5 - Складові трирівневої архітектури на базі обраних інструментів

## Розділ 3. Опис реалізації застосування

### 3.1 Аналіз технічного завдання

Згідно з темою курсової роботи, необхідно реалізувати мобільний застосунок для репетиторів та учнів, котрий би задовольнив вимоги з організації їхньої співпраці та, за можливості, покращив досвід взаємодії між ними. Одним із найголовніших аспектів такого застосування має бути щось, що буде спроможним змушувати користувачів запускати його на своєму пристрої дещо частіше, ніж звичайний довідник з інформацією про людей. Адже, як вже було визначено раніше, суттєвим недоліком деяких сервісів для репетиторів є те, що вони не давали жодного функціоналу, окрім як пошуку контактної інформації репетиторів. Саме тому, спираючись на приклад раніше згаданого додатку для організації репетиторства Wyzant, вирішено реалізувати функціонал, котрий би залучав більш активну участь між обома сторонами освітнього процесу. Відповідно до загального аналізу предметної області, слід виокремити функціонал додатку, котрий би дійсно був легким та корисним у використанні для всіх користувачів.

Тож наш застосунок має передусім пропонувати такі можливості:

- розподіл функціоналу відповідно до обраної ролі учня або викладача під час реєстрації нового акаунту в нашому сервісі;
- можливість пошуку будь-яких зареєстрованих користувачів;
- функція додавання “в друзі” як зі сторони репетитора, так і учня, котра обов’язкова має бути двосторонньо підтверджена;
- можливість для учня залишати відгук на послуги, що були надані його викладачем, але за умови, що він ще є “в друзях” з репетитором, для запобігання неправдивих оцінок та зловживань;
- спроможність легко створювати пропозиції дисциплін з вказанням їхніх назв, описів та відповідних цін;

- створення зручних планувань майбутніх уроків із вказанням додаткових відомостей або завдань для учнів зі сторони викладачів, за умови, що вони двосторонньо “в друзях”;
- реалізація “двобічних” нагадувань у відповідності до наближення запланованих уроків.

Отже, нам необхідно втілити в життя перерахований функціонал задля забезпечення потреб цільової аудиторії нашого сервісу.

## 3.2 Реалізація серверної частини застосування

### 3.2.1 Огляд архітектури серверної частини

Під час реалізації серверної частини нашого застосування будемо передусім спиратися на принципи архітектури REST. Для кращого розуміння, слід визначити деякі основні аспекти її фундаменту, які часто виокремлюють.

По-перше, це розподіл відповідальності (від англ. *separation of concerns*) між обов’язками серверу та клієнту, що до нього звертається. Сенс цієї властивості полягає в тому, що фронтенд (від англ. *frontend*) повинен відповідати лише за відображення та обробку дій користувача в інтерфейсі з відповідними запитами до серверу, а бекенд повинен перейматися за обробку таких запитів із подальшим збереженням даних та втіленням головної логіки сервісу. Таким чином, їхні “сфери впливу” не повинні перетинатися[43].

По-друге, це безстановість (від англ. *stateless*) взаємодії між сервером та клієнтом. Сенс такої особливості вказує на те, що бекенд не повинен зберігати в собі жодну додаткову інформацію під час своєї роботи для того, аби належним чином обробляти запити з боку фронтенду. Таким чином, під час кожного звернення до серверу клієнт надсилає достатньо даних для того, аби той відповідно до їхнього опису успішно сформував відповідь без зайвих витрат ресурсів на допоміжну інформацію[43].

По-третє, це багат шаровість (від англ. *layered*) бекенду. Вона полягає в тому, що сервер має бути складений із окремих шарів, котрі у відповідній

послідовності мають спілкуватися один з одним, передаючи кожному наступному оброблену частину даних. Саме це дозволяє зручніше організувати архітектуру сервера, уникаючи зайвої нагромадженості коду[43].

Врахування вище згаданих особливостей під час написання серверної частини дозволить належними чином структурувати проект і уникнути зайвих труднощів як під час його розробки, так і майбутньої підтримки.

Тож перейдемо до безпосередньої реалізації серверної частини в середовищі Node.js у поєднанні з фреймворком Express. Загальна структура проекту, що зображена на рисунку 6, організована за рекомендаціями впровадження розподіленої архітектури у відповідності до принципів REST.

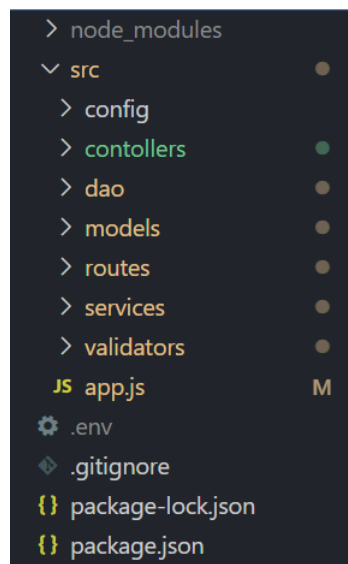


Рисунок 6 - Структура проекту серверної частини

Визначимо призначення кожної частини бекенду на Node.js, що знаходиться у папці `src` проекту на вище вказаній структурі:

- `app.js` - це головна точка входу (від англ. *entry-point*) нашого застосування, де міститься код з визначенням серверу на базі фреймворку Express. Тут насамперед вказано проміжні обробники, котрі дозволяють серверу сприймати та перетворювати тіло отриманих запитів у JSON, також визначено головні шляхи маршрутизації, за котрими будуть здійснюватися звернення, та порт,

на якому відбувається прослуховування сервером запитів (див. додаток В).

- `config` - це місце розміщення всіх конфігурацій під'єднання до використовуваних баз даних.
- `dao` - це точка розміщення всіх файлів, у котрих існує код, що відповідає за роботу шару доступу до даних (від англ. `data access layer`) в БД. Там розміщені імплементації інтерфейсу DAO, котрий реалізує як основні методи роботи з даними CRUD, так і додаткові, що необхідні для предметної області. Тож для кожної сутності, що існує у вигляді таблиці в MySQL або ж у вигляді вузла в Neo4j, існують реалізовані методи для роботи з ними.
- `models` - це розташування класів моделей сутностей, що зберігаються в базах даних. Там визначені поля у відповідності до їхніх властивостей та основні методи для роботи з ними.
- `services` - це місце розміщення основного коду, що відповідає зв'язуючому шару бізнес-логіки застосування, котре відіграє роль містка між шарами доступу до даних та контролерів (від англ. `controllers` - керувальники). Тобто певний метод контролеру викликає такий сервіс, а він, у свою чергу, викликає операції потрібного DAO. Таким чином, у цій папці розташовані всі головні методи, що пов'язані з обробкою даних перед тим, як передати їх до наступного шару.
- `validators` - містить у собі файли, у котрих визначена логіка перевірки даних на відповідність до властивостей їхніх сутностей у БД.
- `routes` - це місце розміщення всіх можливих шляхів маршрутизації нашого застосування з реалізацією обробки звернень, що за ними здійснюються.
- `controllers` - розташування всіх методів, що використовуються для перевірки та передачі отриманих даних зі сторони користувача до наступного шару сервісів, а також для формування відповіді для

клієнту. Вони викликаються безпосередньо в середині методу-обробника запиту, що був здійснений зі сторони користувача відповідно до визначеного маршруту у файлі папки routes.

### 3.2.2 Пояснення реалізації багатошаровості серверної частини на прикладі

Для прикладу скористаємося процесом реєстрації нового облікового запису. Він починається з клієнтського звернення до адреси “/reg\_auth/registrate”, котру перехопить потрібний метод POST фреймворку Express. Під час обробки цього запиту відбувається виклик відповідного методу “registrateUser” контролера AuthRegController (див. Рисунок 7).

```
src > routes > JS authRegRoutes.js > ...
1  const authRegRouter = require("express").Router();
2  const AuthRegController = require("../controllers/authRegController");
3
4  authRegRouter.post("/registrate", (req, res) => {
5    AuthRegController.registrateUser(req.body, res);
6  });
7
```

Рисунок 7 - Визначення шляху “/registrate”

Уже в ньому здійснюється створення унікального ідентифікатора за допомогою пакету uuid[44] для нового акаунту та екземпляру класу UserModel за отриманими полями від клієнту. Тут же відбувається й перевірка даних, використовуючи розширення joi, що були надіслані серверу відповідно до визначених обмежень для акаунту у UserValidator. Пакет joi[45] є спеціальним валідатором (від англ. validator – перевіряльник) полів JavaScript об’єктів, який дозволяє створювати перевірочні схеми, у яких можна легко встановлювати необхідні для контролю властивостей межі (див. додаток Г).

Тільки після того, як вся інформація задовольняє вимоги нашої сутності в БД, відбувається виклик методу “registrate” із сервісу authRegService. Відповідно до результату його виконання, буде визначено, яке повідомлення слід повертати клієнтській частині (див. рисунок 8).

```

registrateUser: async function (data, res) {
  const id = uuidv4();
  data.position = Boolean(parseInt(data.position));
  const user = new UserModel(
    id,
    data.nickname,
    data.name,
    data.surname,
    data.email,
    data.password,
    data.position,
    data.about
  );
  try {
    await UserRegistrationValidator.validateAsync(user);
  } catch (err) {
    return res.json({ success: false, msg: err.details[0].message });
  }
  const registrationSuccess = await authRegService.registrate(user);
  registrationSuccess
    ? res.json({ success: true, msg: "Registration success!" })
    : res.json({ success: false, msg: "Registration error!" });
},

```

Рисунок 8 - Метод-контролер для реєстрації користувача

У самому методі authRegService вже за зібраними та перевіреними даними у попередньому шарі починається шифрування паролю нового облікового запису, залучаючи бібліотеку bcrypt[46]. Вона використовує для криптування відомий алгоритм Blowfish, що, за деякими поданнями, пропонує одні з найшвидших опрацювань інформації, що визначаються гарним показником випадковості кінцевого результату [47]. Тож використання пакету bcrypt неабияк задовольняє потребу у реалізації надійного шифрування паролей для подальшого їхнього збереження у такому вигляді. Далі ж відбувається послідовний виклик методів DAO для збереження користувацьких даних у БД (див. рисунок 9).

```

exports.registrate = async function (user) {
  crypt
    .hash(user.password, saltingRounds)
    .then(async function (hashed) {
      user.password = hashed;
      const mysqlUser = await mysqlUserDAO.create(user);
      const neo4jUser = await neo4jUserDAO.create(user);
      return mysqlUser && neo4jUser;
    })
    .catch(function (err) {
      console.error(err.message);
    });
};

```

Рисунок 9 - Метод “registrate” сервісу AuthRegService

Слід зауважити, що у той час, коли для MySQL ми додаємо повний рядок з інформацією про акаунт відповідно до визначених атрибутів схеми таблиці users, то в графовій Neo4j ми створюємо вузол лише з ідентифікатора, що ми отримали раніше, та ролі, що вказав користувач. Не варто знову повторювати всі відомості про акаунт у двох місцях одночасно, адже це даремно витрачатиме ресурси для обох БД. Дані в Neo4j нам будуть потрібні лише для організації зв'язків між профілями користувачів різних ролей (див. додаток Г), тому немає необхідності дублювати структуровану інформацію, котру ми можемо отримати за допомогою унікального ідентифікатора з реляційної БД.

Отже, маємо реалізовану архітектуру бекенду згідно з принципом розподілу відповідальності між його частинами. Це дозволило створити структуровану систему з шарів, кожний із яких відповідає за визначені обов'язки. Внаслідок того, що відбувається уникання зайвого перетину сфер їхньої відповідальності, вдається оминати нагромадження та забезпечити більшу читабельність коду.

### 3.2.3 Створення авторизації

Авторизація дій користувача на стороні сервера буде реалізована шляхом використання JWT-токенів (від англ. token - жетон). Це спеціальна стрічка, що вміщує в себе закодовану JSON інформацію. Її особливість полягає в тому, що вона дозволяє шляхом використання ключів-підписів визначати те, чи можна довіряти інформації, яка в ній знаходиться. Цей токен складається з трьох зашифрованих частин: Header, Payload та Signature, які, в свою чергу, розділені між собою крапками. Перша частина містить в собі інформацію про тип жетону та алгоритм кодування. Друга - основні дані, що надсилаються у ньому, а третя - підпис, за допомогою якого можна визначити те, чи не була внутрішня інформація токена якось змінена[48]. Таким чином, використання JWT може полегшити впровадження авторизації, адже такі токени можуть вміщувати в собі всю необхідну інформацію для цього. Більше того, ця інформація підписана



ключем, який може створити сам розробник. Така стрічка має надаватися користувачеві кожний раз, коли той успішно проходить автентифікацію.

Хоч надані користувачам токени і надійно захищені, нічого не може бути вічним у протистоянні зі зловмисниками. Саме тому існує практика їхнього навмисного “зістарювання” шляхом вказання у Payload часу, через який цей жетон перестане працювати (див. рисунок 10). Тож кожний раз, коли буде застарівати минулий JWT, необхідно буде створювати повністю новий, але перед цим необхідно переконатися, що створювач є тією самою персоною, що й раніше. Для вирішення цієї проблеми використовують допоміжні токени, котрі мають надійно зберігатися на стороні клієнту і при цьому мати більший час придатності.

```
function createTokens(user) {  
  const mainToken = jwt.sign(  
    { nickname: user.nickname, id: user.id },  
    mainSecret,  
    { expiresIn: "2m" }  
  );  
  const refreshToken = jwt.sign(  
    { nickname: user.nickname, id: user.id },  
    refreshSecret,  
    { expiresIn: "1h" }  
  );  
  return { mainToken: mainToken, refreshToken: refreshToken };  
}
```

Рисунок 10 - Метод створення tokenів за властивостями користувача

Таким чином, залучення JWT може повною мірою задовольнити потреби реалізації авторизації на стороні сервера. До того ж, використання tokenів відповідно до існуючих практик забезпечує не тільки надійну перевірку кожного запиту від клієнта, а й принцип безстановості REST архітектури. Адже сервер не зберігає жодної додаткової інформації про кожного користувача, аби забезпечувати перевірку його дій.

На противагу методам, що пов’язані із реєстрацією та аутентифікацією, усі інші, що доступні користувачеві, обов’язково мають проходити перевірку на “дозволеність”. До того ж, ця валідація має відбуватися до моменту початку обробки запиту сервером, адже інакше в ній не буде жодної користі. Внаслідок

того, що надає нам фреймворк Express у викликах його методів у відповідності до HTTP запитів, можна із легкістю реалізувати попередню авторизацію дій клієнта. Завдяки можливості введення додаткових проміжних обробників під назвою `middleware` (від англ. проміжне забезпечення), можна реалізувати спеціальний метод авторизації та ввести його у вигляді такого обробника попередньо до початку опрацювання отриманих даних у зверненні (див. рисунок 11).

```
planRouter.post("/add_plan", authHelper.authorize, (req, res) => {
  planController.addPlan(req.body, res);
});
```

Рисунок 11 - Приклад розміщення проміжної функції авторизації

### 3.3 Реалізація клієнтської частини застосування

#### 3.3.1 Огляд архітектури клієнтського застосування

Під час розробки на Flutter можна обирати різні підходи до реалізації архітектури застосунку, хоч і прямої рекомендації від Google з приводу “правильного” вибору не існує. Тож вирішення даного питання лежить на плечах програмістів. На щастя, сьогодні існує достатня кількість практик того, як слід структурувати клієнтську частину. Як наслідок, під час реалізації був обраний один з досить поширених у використанні шаблонів проектування, а саме MVC. Він являється одним із найперших запропонованих підходів до розбиття рівня представлення від інших. Тож головною метою слідування принципам такого патерну проектування (від англ. pattern – шаблон), як і багатьох інших, є передусім розподіл системи на окремі компоненти, котрі б дозволили розвантажити її нагромадженість.

Основний аспект роботи MVC полягає у розбитті структури застосування на три частини: Model, View та Controller відповідно. Завданням Model є зберігання даних об’єкту. На View лежить відображення користувацького інтерфейсу відповідно до конкретного стану існуючих даних, із котрим, у свою чергу, на пряму взаємодіє користувач. У Controller же завдання полягає у

керуванні обома попередніми компонентами, адже саме він приймає результати взаємодії з View, обробляє їх потрібним чином та змінює стан даних у Model. Після того, як необхідні дані були змінені, Controller має знову “сповістити” про це View, аби ця частина могла відобразити актуальну на конкретний момент інформацію[49].

Отже, MVC дозволяє організувати розробку проекту таким чином, щоб було легше створювати й підтримувати наявний код, особливо коли мова йде про команду розробників. Адже таке розбиття дозволить окремим фахівцям займатися відповідним компонентом, не втручаючись у зміни іншого. Більше того, ізольованість кожного компоненту у разі доречної реалізації може послугувати неабияким помічником у інших проектах, оскільки буде можливість використовувати, наприклад, раніше створенні моделі та контролери для використання з іншими представленнями.

Для того, аби було легше відображати зміни елементів View, вирішено скористатися певною бібліотекою для керування станом компонентів каркасу Flutter, що має назву Provider[50]. Як було розглянуто раніше весь користувацький інтерфейс, що написаний на цьому фреймворку, складається із дерев віджетів. Деякі ж з них можуть зазнавати перетворень внаслідок зміни опису їхніх станів під час реагування UI на взаємодію зі сторони користувача.

Стан - це певна інформація, котра зчитується елементом під час його побудови і може змінюватися час від часу впродовж його існування. У Flutter для зберігання такої інформації для кожного віджету створюється спеціальний об'єкт State. Оновлювати цей об'єкт можна шляхом виклику для віджета методу “setState”, у якому ми можемо вказати, які саме властивості мають змінитися, спровокувавши перемалювання потрібних компонентів. Постійне використання такого методу може бути досить незручним, адже для того, аби оновити стан кількох віджетів одночасно, ми маємо викликати цей метод для кожного з них. Більше того, це буде провокувати перебудову всього віджету протягом його існування. Тому пропонують користуватися іншими підходами, зокрема шляхом підписання на певні об'єкти, що можуть сповіщати віджети про якісь зміни.

Кожне таке сповіщення перемальовуватиме лише ті елементи, що й використовують нові дані[51]. Саме тому й доречно використати бібліотеку Provider, що й базується на другому підході. Він дозволить зручно “прослуховувати” зміну станів крізь весь інтерфейс додатку, адже слідкування за кожним окремим віджетом може призвести до зайвої плутанини, а деякі функції, зокрема зміна теми кольорової схеми всього застосунку, взагалі будуть надто складними у втіленні та затратними для застосування.

Для розуміння того, для чого потрібні динамічні віджети та керування їхнім станом, можна розглянути приклад на рівні підтвердження “запрошення до друзів”, що так часто зустрічається у соціальних мережах. Тож уявімо, що, коли отримано новий такий запит, користувач переходить на екран його підтвердження і йому пропонується відповідна кнопка для цього. Як тільки, відбувається натискання такої кнопки, то інтерфейс цього запрошення має відобразити якісь перетворення для сповіщення, що була здійснена певна дія. Аби втілити таке в життя, нам необхідно вплинути на стан динамічних віджетів цього екрану, які й допоможуть відобразити необхідні для розуміння користувача зміни інтерфейсу. Дуже часто деякі такі дії користувача мають одразу відобразитися не лише на певний віджет, а й на весь застосунок.

### 3.3.2 Огляд реалізованої архітектури клієнтської частини

Файлова структура проекту відповідає головним компонентам MVC, а також включає в себе папки з реалізованими в них допоміжними сервісами та власними віджетами (див. рисунок 13). Файл main.dart відповідає за головну точку входу нашого застосунку на Flutter.

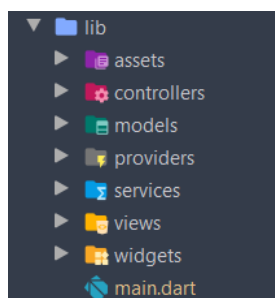


Рисунок 13 - Структура проекту клієнтської частини

Для демонстрації реалізованої архітектури на базі шаблону MVC буде використано процес проходження автентифікації. Нехай користувач уже ввів необхідні дані для свого облікового запису і натискає на кнопку для здійснення входу в акаунт. Натискаючи на неї, користувач викликає метод контролеру “loginUser”. Сам метод є асинхронним, оскільки в ньому викликаються також асинхронні методи, результат яких нам необхідно отримати для майбутнього встановлення значень у моделі користувача (див. рисунок 12).

```
Future<bool> loginUser(context, nickname, password) async {  
  var bodyMap = await AuthService.instance.login(nickname, password);  
  if (!bodyMap['success']) return false;  
  UserModel user = UserModel.fromJson(bodyMap);  
  if (user != null) {  
    await Provider.of<UserProvider>(context, listen: false)  
      .setUserWithTokens(bodyMap['mainToken'], bodyMap['refreshToken'], user);  
    return true;  
  }  
  return false;  
}
```

Рисунок 12 - Метод-контролер автентифікації користувача

Тут же відбувається поступовий виклик функцій необхідних сервісів, що роблять потрібні типи HTTP запитів до серверу за вказаними адресами. Після успішного отримання результатів виконання операції автентифікації, цей метод здійснює збереження необхідних даних у UserModel.

Як тільки всю потрібну інформацію було збережено, контролер звертається до виклику методу “setUserWithTokens” класу UserProvider, на який було підписано наше застосування ще у методі main. Всередині нього відбувається виклик функції “NotifyListeners”, який сповіщає компоненти інтерфейсу нашого додатку про те, що всі дані автентифікованого користувача було отримано й збережено.

### 3.3.3 Робота з даними для авторизації дій користувачів

Як було раніше згадано у підрозділі реалізації серверної частини застосування, для авторизації дій користувачів на бекенді використовується JWT. Щоб кожне звернення, котре здійснює користувач, було розглянуте сервером, а не відкинуте як потенційно небезпечне, треба під час формування HTTP запиту у частині його заголовку, що дозволяє надсилати в собі додаткову нечутливу до регістру інформацію[52], вказувати певний токен. Такий зашифрований жетон видається клієнтській частині кожний раз, коли користувач успішно проходить аутентифікацію. Насправді, як уже було сказано, цей токен має проміжок часу, під час якого він може бути використаний для роботи з сервером. Як наслідок, користувачу доводилося б кожного разу, перед тим як здійснити якусь дію у додатку, проходити процес перевірки особистості, але це авжеж не дуже зручно. Через це необхідно отриманий токен протягом його придатності десь безпечно зберігати на стороні клієнту.

Завдяки тому, що клієнтська частина реалізовується у вигляді мобільного застосування, можна скористатися створенням на пристрої спеціальних сховищ, на кшталт KeyStore, що в свою чергу матимуть досить надійне шифрування, ключ якого буде дуже важко дістати навіть самому користувачу[53]. Вони чудово підійдуть у ролі безпечного місця для зберігання стрічок JWT, адже повністю задовольняють вимоги для такого роду даних. Створити таке сховище за допомогою Flutter досить легко, бо існує спеціальне для цього розширення FlutterSecureStorage від спільноти. Воно надає відповідне API, за допомогою якого можна з легкістю створювати сховище, записувати та зчитувати із нього дані. Також FlutterSecureStorage є кросплатформним розширенням, завдяки чому можна реалізувати таке сховище на кількох платформах. Для кожної ОС це розширення використовує власний підхід шифрування, наприклад, для iOS використовується сервіс Keychain, а для Android згаданий KeyStore[54]. Таким чином, спільнота цього фреймворку допомогла зручно вирішити проблему зберігання чутливих даних для багатьох розробників. Отже, реалізувавши

зручний клас з методами для роботи з внутрішнім сховищем системи за допомогою використання API `FlutterSecureStorage`, можемо здійснювати належне збереження стрічок JWT. Після того, як токени будуть успішно збережені, ми можемо зчитувати їх із зашифрованого сховища для надсилання до серверу у будь-якій точці застосування (див. рисунок 14).

```
static Future<Map<String, String>> setHeaderTokens() async {
    var mainToken = await StorageService.instance.fetchFromUserStorage('mainToken');
    var refreshToken = await StorageService.instance.fetchFromUserStorage('refreshToken');
    return {"authorization": "Bearer $mainToken", "reauthorization": "Bearer $refreshToken"};
}

static Future<dynamic> postResponseBy(String fullURL, Map<String, String> postBody) async {
    var response = await http.post(fullURL, headers: await setHeaderTokens(), body: postBody);
    return response;
}
```

Рисунок 14 - Метод зчитування токенів зі сховища

### 3.3.4 Реалізація системи нагадувань

Для реалізації системи нагадувань було використано бібліотеку `FlutterLocalNotifications`. Вона також підтримує кілька мобільних ОС і використовує індивідуальні служби формування нагадувань для кожної. Саме тому для коректного створення таких повідомлень необхідно окремо ініціалізувати налаштування для Android та iOS[55] (див. рисунок 15).

```
final AndroidInitializationSettings initializationSettingsAndroid = AndroidInitializationSettings('flutter');

final IOSInitializationSettings initializationSettingsIOS = IOSInitializationSettings(
    requestSoundPermission: true, requestBadgePermission: false,
    requestAlertPermission: false, onDidReceiveLocalNotification:
        (int id, String title, String body, String payload) async {});

final InitializationSettings initializationSettings =
    InitializationSettings(android: initializationSettingsAndroid, iOS: initializationSettingsIOS, macOS: null);
```

Рисунок 15 - Методи ініціалізації налаштувань нагадувань для ОС

Після цього можна імплементувати (від англ. *implement* – впровадити) метод, котрий дозволяє спланувати нагадування відповідно до вказаного часу, вписуючи необхідну інформацію для відображення в його майбутній картці. Саме ця картка й з’являтиметься на пристрої користувача у потрібний час. Також

можна налаштувати звук та бейдж повідомлення, визначаючи специфічні деталі для нагадування на кожній системі.

Документація даної бібліотеки дозволяє з легкістю опанувати те, як правильно використовувати її можливості для створення простих, але досить гарних нагадувань для Android та iOS. Це неабияк заощаджує час у порівнянні з вивченням нативної імплементації, котру, за потреби, розробники також можуть реалізувати.

### 3.4 Тестовий огляд роботи реалізованого застосунку

Таким чином, розроблено всі ключові частини нашого застосування для його повноцінного функціонування. Для того, аби здійснити тестовий огляд того, що пропонує створений додаток, варто почати відтворення досвіду реального користувача, який вирішив його встановити в себе на пристрої. Почнемо з початкового екрану-привітання, котре обов'язково зустрічатиме користувачів при першому запуску мобільного застосунку.

Тут на вибір є дві можливості, кожна з яких визначає шлях як для зареєстрованих, так і ні людей. Нехай користувач не має акаунту, тому спочатку йому необхідно звернутися до вікна реєстрації. Заповнюючи форму необхідними даними, в нього також буде можливість обрати те, яку з ролей він прагне посідати - учнівську або ж викладацьку. Після реєстрації новий користувач буде одразу перенаправлений на вікно входу в акаунт (див. рисунок 16). Після успішного проходження автентифікації, його буде направлено на сторінку свого профілю, де, за бажання, він може додати більше інформації про себе, котру бачитимуть інші (див. рисунок 17).



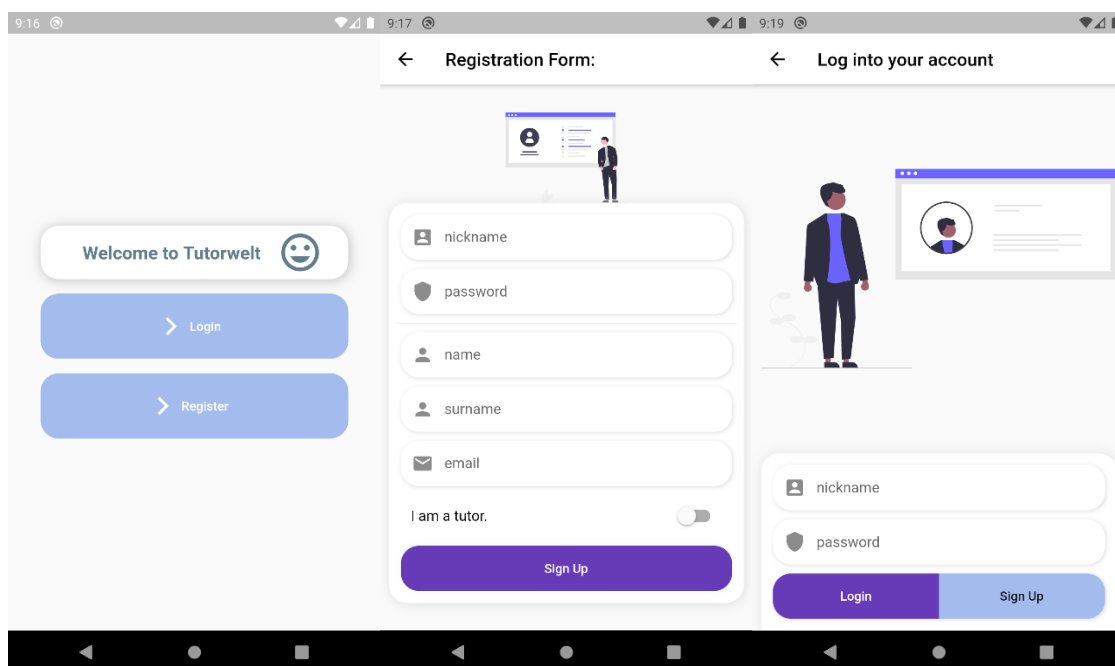


Рисунок 16 - Екран-привітання, форма реєстрації та форма автентифікації

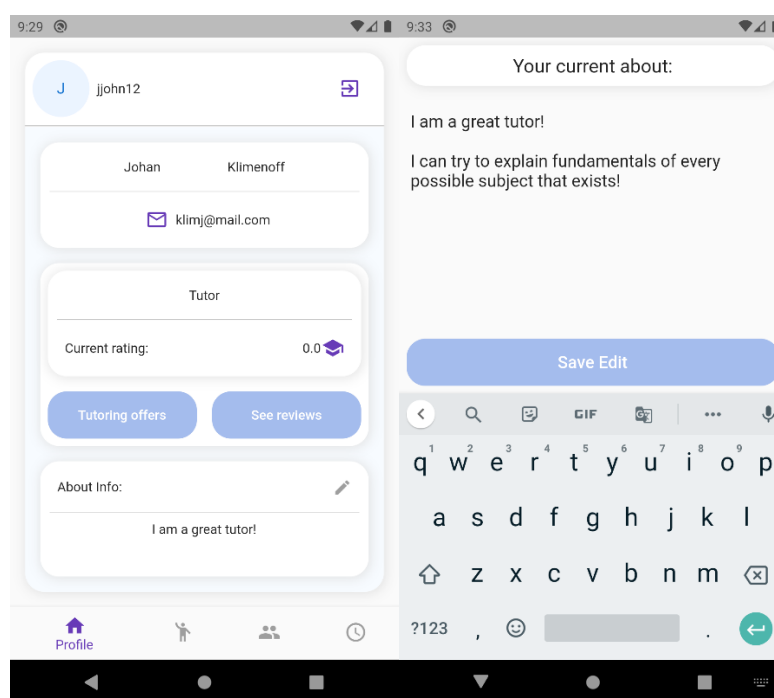


Рисунок 17 - Профіль користувача та вікно редагування інформації

Тож за умови, що користувач створив акаунт у ролі репетитора, на екрані профілю в нього буде спроможність перейти до створення пропозицій дисциплін з чітким визначенням їхніх назв, описів та відповідних цін (див. рисунок 18). В учня відповідної можливості існувати не буде.

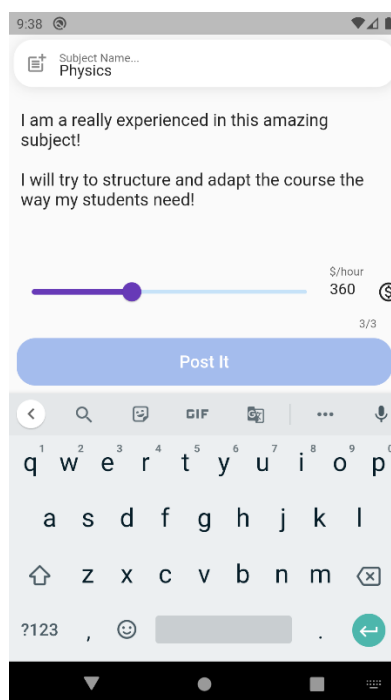


Рисунок 18 - Екран створення дисципліни-пропозиції

Після цього користувач передусім захоче якось зв'язатися або з іншим учнем, або ж з іншим викладачем. Для цього йому необхідно перейти до розділу “People”. У ньому буде можливість здійснити пошук за трьома категоріями користувачів, дві з яких відповідають вказаним ролям, а одна розміщує в собі вже існуючих “знайомих”. Отже, здійснивши за певним набором літер пошук, наприклад, у підрозділі учнів, потрібного користувача, у інтерфейсі будуть відображені компактні картки профілів, у яких буде міститися основна інформація. Якщо ж людина прагне створити зв'язок зі знайденим акаунтом, йому необхідно натиснути на цю картку. Після цього відкриється екран повного профілю іншого користувача, на котрому буде можливість натиснути кнопку “Init Connection”, що надішле потрібний запит на додавання до “друзів”. Про успішність ініціації зв'язку повідомить спеціальний індикатор з текстом, що відображатиметься на місці натиснутої раніше кнопки (див. рисунок 19).

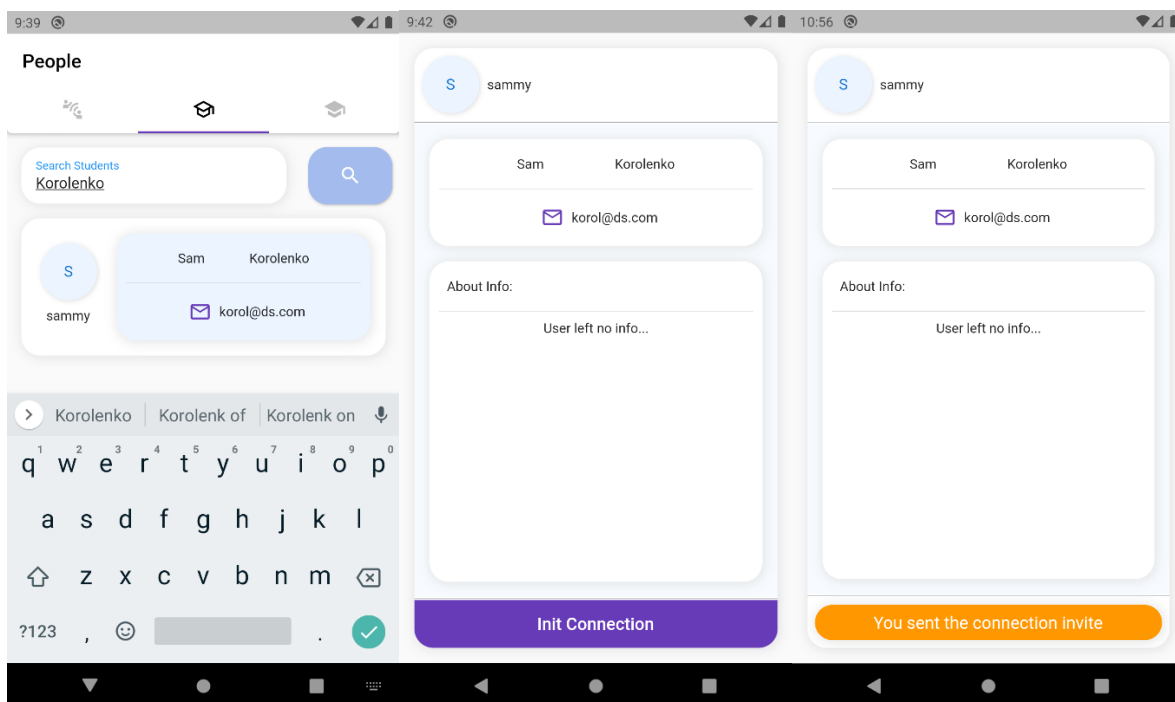


Рисунок 19 - Екран пошуку користувача та екран з'єднання з ним

Після цього можемо аутентифікуватися зі сторони користувача, котрому було надіслано запит вище. У нього в панелі навігації на піктограмі “Requests” з’явиться мітка, яка вказуватиме на отримання запиту від когось (див. рисунок 20). Під час переходу до відповідного підрозділу в нього буде можливість здійснити підтвердження цього запиту.

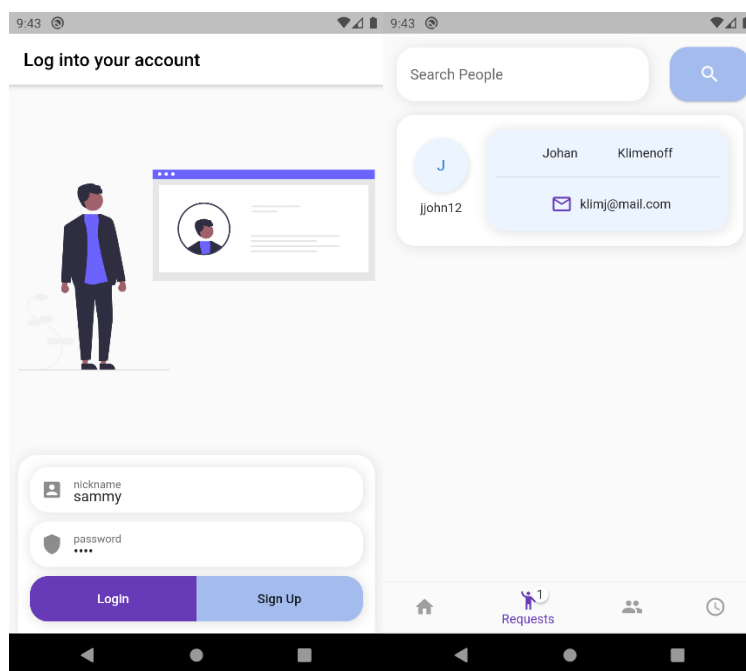


Рисунок 20 - Екран входження в інший акаунт та розділ отриманих запрошень

Після цього користувачі з'являться один для одного в розділі “друзів”. Наявність зв'язку принаймні з однією особистістю дозволить створювати планування зустрічі, якщо людина має роль викладача. На екрані створення такого плану в репетитора буде можливість вказати відповідний час, учня, розділ для написання повідомлення для нього та певні нотатки для себе (див. рисунок 21). Коли зустріч почне наближатися, то в користувача з'являтимуться картки-нагадування з указанням того, скільки часу до зустрічі залишилося та з ким вона запланована (див. додаток Д).

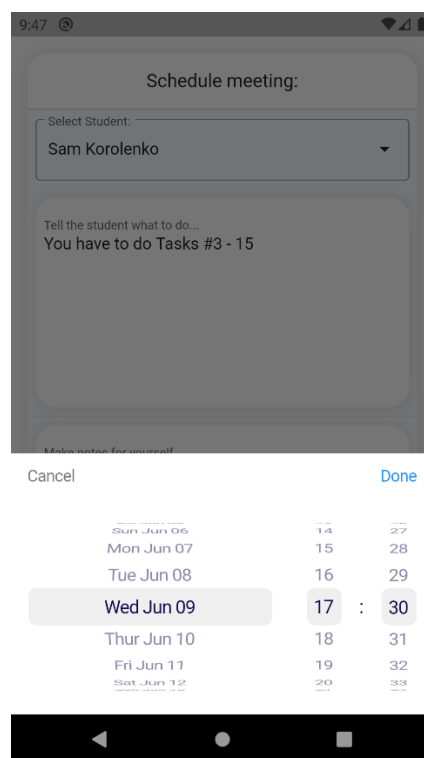


Рисунок 21 - Створення плану наступної зустрічі-уроку

Також слід згадати й про можливість залишати відгуки на репетиторів, за умови, що користувач є з ними у зв'язку. Це зроблено для того, аби уникнути несправжніх відгуків зі сторони людей, що можливо ніколи й не мали справи з викладачем. На екрані створення відгуку, крім написання тексту, надається можливість поставити оцінку за п'ятибальною шкалою (див. рисунок 22). Кожна надана оцінка враховуватиметься в середньому рейтингу репетитора, який можна потім побачити на його профілі.

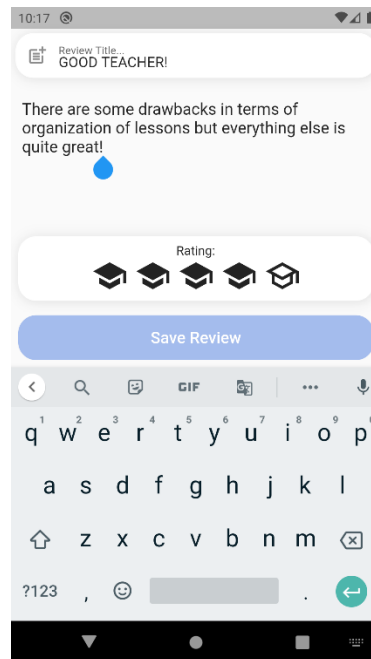


Рисунок 22 - Екран написання анонімного відгуку на репетитора

Отже, внаслідок відтворення дій реального сценарію користування додатком, було здійснено огляд всіх можливостей додатку та його інтерфейсу, що був створений за допомогою фреймворку Flutter.

### 3.5 Висновок до розділу 3

У даному завершальному розділі курсової роботи було здійснено опис повної реалізації додатку для репетиторів на базі раніше обраних інструментів. Було проведено аналіз технічного завдання з визначенням необхідного функціоналу відповідно до предметної області, а також розглянуто архітектурні рішення, які були використані під час реалізації кожної частини клієнт-серверного застосунку.

Також був зроблений опис імплементації головних принципів обраних підходів із наданням скріншотів (від англ. screenshot - зображень екрану) коду, крім того, розглянуто допоміжні розширення та бібліотеки. У кінці було покроково відтворено сценарій використання готового застосування користувачем із супроводом у вигляді зображень екранів. Продемонстровано кожен реалізовану можливість згідно з раніше розглянутими у технічному завданні пунктами.

## Висновки

Підсумовуючи, у результаті виконання курсової роботи було створено клієнт-серверний застосунок для сучасних телефонів, котрий спрямований на задоволення потреб репетиторської діяльності. Його було успішно розроблено на базі обраних технологій: Flutter та Node.js, а також їхніх розширень та додаткових бібліотек. Під час розробки були описані всі залучені архітектурні принципи, зокрема розглянуто та втілено деякі з популярних рішень для розробки обох частин системи: клієнту та серверу. У відповідності до проаналізованої предметної області, був втілений функціонал, який стане в нагоді цільовій аудиторії. Також було продемонстровано можливий сценарій використання готового застосунку користувачем.

Крім того, у результаті роботи над створенням додатку були визначені перспективи подальшого розвитку та покращення його функціональної складової. Зокрема в ньому можна забезпечити можливість оплати репетиторських послуг безпосередньо у застосуванні, а також додати рекомендаційний двигун на базі визначення місцезнаходження користувачів, аби учням було легше знаходити репетиторів ближчих до себе. Більше того, технології та інструменти, які були використані під час реалізації застосування, дозволяють втілити різні можливості, які не обмежуються зазначеними. До того ж, якщо з часом кількість послуг, які пропонуватиме застосунок, стане надто великою, то буде доречно модернізувати серверну частину застосування шляхом її структурованого розбиття на надання окремих сервісів, аби не перевантажувати всім один сервер. Як наслідок, маємо мобільне застосування, межі покращення та розширення якого обмежуються лише фантазією та часом.

## Список Використаних Джерел

1. Serpstat: [Електронний ресурс]. 2021.  
<https://serpstat.com/uk/> (дата звернення: 03.05.2021).
2. Wyzant – Find Expert Tutors // Google Play: [Електронний ресурс]. 2021.  
<https://play.google.com/store/apps/details?id=com.wyzant.studentapp&hl=uk&gl=US> (дата звернення: 03.05.2021).
3. About // Wyzant: [Електронний ресурс]. 2021.  
<https://www.wyzant.com/about> (дата звернення: 03.05.2021)
4. Rogers B. Creating an Online Marketplace for Tutors and Students // Forbes: [Електронний ресурс]. 2014  
<https://www.forbes.com/sites/brucerogers/2014/09/17/creating-an-online-marketplace-for-tutors-and-students/?sh=5b9ea55e6c0e> (дата звернення: 03.05.2021)
5. IOS Apple App Store Statistics And Trends // 42 Matters: [Електронний ресурс]. 2021.  
<https://42matters.com/ios-apple-app-store-statistics-and-trends> (дата звернення: 08.04.2021)
6. Development of Native Mobile Application Using Android Studio for Cabs and Some Glimpse of Cross Platform Apps // International Journal of Applied Engineering Research. 2018, вип. 16 Т. 13. с. 12527. [Електронний ресурс]  
[http://www.ripublication.com/ijaer18/ijaerv13n16\\_17.pdf](http://www.ripublication.com/ijaer18/ijaerv13n16_17.pdf) (дата звернення: 08.04.2021)
7. W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, i A. M. Wahba, «Taxonomy of Cross-Platform Mobile Applications Development Approaches» // Ain Shams Engineering Journal, 2017, вип. 8, Т. 2, с. 164.[Електронний ресурс]  
<http://dx.doi.org/10.1016/j.asej.2015.08.004> (дата звернення: 09.04.2021)
8. Native vs Cross-Platform App Development // LinkedIn – Pulse. 2020: [Електронний ресурс]

- <https://www.linkedin.com/pulse/native-vs-cross-platform-app-development-joe-neal?articleId=6630821857580851201> (дата звернення: 09.04.2021).
9. Web.Dev – What are Progressive Web Apps? : [Електронний ресурс]  
<https://web.dev/what-are-pwas/> (дата звернення: 09.05.2021).
  - 10.MDN Web Docs – Service Worker API : [Електронний ресурс]  
[https://developer.mozilla.org/en-US/docs/Web/API/Service\\_Worker\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API)  
(дата звернення: 09.04.2021).
  - 11.ElHady H. Instabug – Blog - Your Guide to Cross-Platform Mobile App Development Tools : [Електронний ресурс] [https://instabug.com/blog/cross-platform-development/?utm\\_source=quora&utm\\_medium=social&utm\\_campaign=hady11&utm\\_content=cross\\_platform\\_development](https://instabug.com/blog/cross-platform-development/?utm_source=quora&utm_medium=social&utm_campaign=hady11&utm_content=cross_platform_development) (дата звернення: 09.04.2021).
  - 12.Android Developers - Building Web apps in WebView : [Електронний ресурс]  
<https://developer.android.com/guide/webapps/webview> (дата звернення: 09.04.2021).
  - 13.Doc Rust - Unsafe Rust : [Електронний ресурс]  
<https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html#using-external-functions-to-call-external-code> (дата звернення: 12.05.2021).
  - 14.Bjørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, i G. Ghinea, «An empirical investigation of performance overhead in cross-platform mobile development frameworks», Empir Software Eng, 2020, вип. 25, Т. 4, С. 3002-3003, 3008 : [Електронний ресурс]  
<http://dx.doi.org/10.1007/s10664-020-09827-6>
  - 15.JetBrains - The State of Developer Ecosystem 2020 : [Електронний ресурс]  
<https://www.jetbrains.com/lp/devecosystem-2020/>
  - 16.JetBrains - The State of Developer Ecosystem 2019 : [Електронний ресурс]  
<https://www.jetbrains.com/lp/devecosystem-2019/>



17. GitHub – Flutter Open Repo : [Электронный ресурс]  
<https://github.com/flutter/flutter> (дата звернения: 17.01.2021).
18. GitHub – React Open Repo : [Электронный ресурс]  
<https://github.com/facebook/react> (дата звернения: 17.01.2021).
19. GitHub – Cordova Open Repo : [Электронный ресурс]  
<https://github.com/apache/cordova-android> (дата звернения: 17.01.2021).
20. Flutter – Dev : [Электронный ресурс]  
<https://flutter.dev/> (дата звернения: 10.04.2021).
21. Pub – Dev : [Электронный ресурс]  
<https://pub.dev/> (дата звернения: 10.04.2021).
22. Fabacher T. Forbes - Why You Should Start Looking At Google's Flutter And Fuchsia Now: [Электронный ресурс]  
<https://www.forbes.com/sites/forbesnycouncil/2018/07/26/why-you-should-start-looking-at-googles-flutter-and-fuchsia-now/?sh=2bda656aa309>  
(дата звернения: 10.02.2021).
23. Flutter – Dev – Showcase : [Электронный ресурс]  
<https://flutter.dev/showcase> (дата звернения: 10.04.2021).
24. Dart – Dev – Overview : [Электронный ресурс]  
<https://dart.dev/overview> (дата звернения: 10.04.2021).
25. Flutter – Dev – Docs : [Электронный ресурс]  
<https://flutter.dev/docs/development/ui/widgets-intro> (дата звернения: 10.04.2021).
26. Hackr.io – Blog – Web Development Frameworks : [Электронный ресурс]  
<https://hackr.io/blog/web-development-frameworks> (дата звернения: 03.05.2021).
27. Spring IO – Docs : [Электронный ресурс]  
<https://spring.io/> (дата звернения: 03.05.2021).
28. Spring IO – Why Spring? : [Электронный ресурс]  
<https://spring.io/why-spring> (дата звернения: 03.05.2021).

- 29.Django Project – Overview : [Электронный ресурс]  
<https://www.djangoproject.com/start/overview/> (дата звернення: 03.05.2021).
- 30.Node.js – About : [Электронный ресурс]  
<https://nodejs.org/en/about/> (дата звернення: 03.05.2021).
- 31.Mohllal K. Medium – Node.js Multithreading? : [Электронный ресурс]  
<https://medium.com/@mohllal/node-js-multithreading-a5cd74958a67> (дата звернення: 03.05.2021).
- 32.Capan T. TopTal – Node.js – Why? : [Электронный ресурс]  
<https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> (дата звернення: 03.05.2021).
33. About // NPM. 2021 : [Электронный ресурс]  
<https://www.npmjs.com/about> (дата звернення: 03.05.2021).
- 34.MDN – Web Docs // Express/Node introduction : [Электронный ресурс]  
[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction) (дата звернення: 03.05.2021).
- 35.Express // NPM : [Электронный ресурс]  
<https://www.npmjs.com/package/express> (дата звернення: 03.05.2021).
- 36.API // Express.js : [Электронный ресурс]  
<https://expressjs.com/en/4x/api.html#app> (дата звернення: 03.05.2021).
- 37.MongoDB - What is a Non-Relational Database? : [Электронный ресурс]  
<https://www.mongodb.com/non-relational-database> (дата звернення: 05.05.2021).
- 38.CodeWave – Insights – When to use MongoDB and Why? : [Электронный ресурс]  
<https://insights.codewave.com/when-to-use-mongodb-and-why/> (дата звернення: 05.05.2021).
- 39.Survey – StackOverflow – Databases : [Электронный ресурс]  
<https://insights.stackoverflow.com/survey/2020#technology-databases-all-respondents4>
- 40.MySQL – Documentation - The Main Features of MySQL : [Электронный ресурс]

<https://dev.mysql.com/doc/refman/8.0/en/features.html> (дата звернення: 05.05.2021).

41.Neo4j – What is Neo4j? : [Електронний ресурс]

<https://neo4j.com/> (дата звернення: 05.05.2021).

42.Haroon Shakirat Oluwatosin IOSR Journal of Computer Engineering (IOSR-JCE) // Client-Server Model, 2014, Т. 16, вип. 1, вер. IX, PP 68-69

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1083.8741&rep=rep1&type=pdf>

43.F. Doglio, Pro REST API Development with Node.js. Apress, 2015, ст. 3-6

<http://dx.doi.org/10.1007/978-1-4842-0917-2>

44.NPM – uuid : [Електронний ресурс]

<https://www.npmjs.com/package/uuid> (дата звернення: 07.05.2021).

45.NPM – joi : [Електронний ресурс]

<https://www.npmjs.com/package/joi> (дата звернення: 07.05.2021).

46.NPM – bcrypt : [Електронний ресурс]

<https://www.npmjs.com/package/bcrypt> (дата звернення: 07.05.2021).

47.P. Patil, P. Narayankar, Narayan D.G., i Meena S.M., «A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish», Procedia Computer Science, вип. 78, с. 617–624, 2016 :

<http://dx.doi.org/10.1016/j.procs.2016.02.108>

48.JWT – Introduction : [Електронний ресурс]

<https://jwt.io/introduction> (дата звернення: 07.05.2021).

49.Lou T. A comparison of Android Native App Architecture: MVC, MVP and MVVM : дис. на здобуття наук. ступеня Major: Service Design and Engineering : 11 : захист 31.10.2016 / наук. кер. Boudewijn F. van Dongen & Heikki Saikonen. Ейндховен, 2016. 45 с : [Електронний ресурс]

[https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou\\_2016.pdf](https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf)

50.Pub Dev – Provider : [Електронний ресурс]

<https://pub.dev/packages/provider> (дата звернення: 07.05.2021).

51.Flutter – Widgets – StatefulWidget : [Электронный ресурс]

<https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html> (дата  
звернення: 07.05.2021).

52.MDN – Web Docs – HTTP – Headers : [Электронный ресурс]

<https://developer.mozilla.org/ru/docs/Web/HTTP/Headers> (дата звернення:  
07.05.2021).

53.Android Developers – Training Article – KeyStore : [Электронный ресурс]

<https://developer.android.com/training/articles/keystore> (дата звернення:  
07.05.2021).

54.Pub Dev – Flutter Secure Storage : [Электронный ресурс]

[https://pub.dev/packages/flutter\\_secure\\_storage](https://pub.dev/packages/flutter_secure_storage) (дата звернення:  
07.05.2021).















55.Pub Dev – Flutter Local Notifications : [Электронный ресурс]

[https://pub.dev/packages/flutter\\_local\\_notifications](https://pub.dev/packages/flutter_local_notifications) (дата звернення:  
07.05.2021).

## Додатки

## Додаток А

## Результати аналізу запитів із ключовим словом репетитор у Serpstat

Ключові слова <b>репетитор</b>									
#	<input type="checkbox"/>	Ключова фраза		Складність ▼	Частотність ↓	Вартість (\$) ▼	Конкуренц., % ▼	Результатів ▼	
1	<input type="checkbox"/>	репетиторы	  ▶	29	4.4K	0,07	40	9.2M	
2	<input type="checkbox"/>	репетитор	  ▶	37	4.4K	0,07	40	9.9M	
3	<input type="checkbox"/>	ваш репетитор	 ▶	23	2.9K	0,08	29	2.2M	
4	<input type="checkbox"/>	репетиторы английский	 ▶	36	2.4K	0,22	71	11.5M	
5	<input type="checkbox"/>	буки репетиторы	  ▶	13	2.4K	0,12	24	22	
6	<input type="checkbox"/>	репетитор английского	 ▶	36	2.4K	0,22	71	17.1M	
7	<input type="checkbox"/>	буки репетитор	 ▶	12	2.4K	0,07	21	20	
8	<input type="checkbox"/>	репетитор по математике	 ▶	18	1.9K	0,21	72	10.4M	
9	<input type="checkbox"/>	репетитор английского киев	 ▶	35	1.6K	0,26	81	67	
10	<input type="checkbox"/>	репетитор по математике киев	  ▶	16	1K	0,24	72	52	

## Додаток Б

## Визначення показнику “частотність” у звіті Serpstat

Частотність ↓		
Частотність фрази в місяць згідно Google Ads (середнє значення за останній рік)		

## Додаток В

Вміст файлу app.js - точки входу серверного застосування

```
src > JS app.js > ...
1  const authRegRouter = require("./routes/authRegRoutes");
2  const userRouter = require("./routes/userRoutes");
3  const reviewsRouter = require("./routes/reviewRoutes");
4  const usersRouter = require("./routes/usersRoutes");
5  const postRouter = require("./routes/postRoutes");
6  const planRouter = require("./routes/planRoutes");
7  const { port } = require("./config");
8
9  const express = require("express");
10 const app = express();
11
12 app.use(express.urlencoded({ extended: false }));
13 app.use(express.json());
14
15 app.use("/reg_auth", authRegRouter);
16 app.use("/user", userRouter);
17 app.use("/review", reviewsRouter);
18 app.use("/users", usersRouter);
19 app.use("/post", postRouter);
20 app.use("/plan", planRouter);
21
22 app.listen(port, () => {
23   console.log("This Express Server is currently UP and Listening...");
24 });
25
```

## Додаток Г

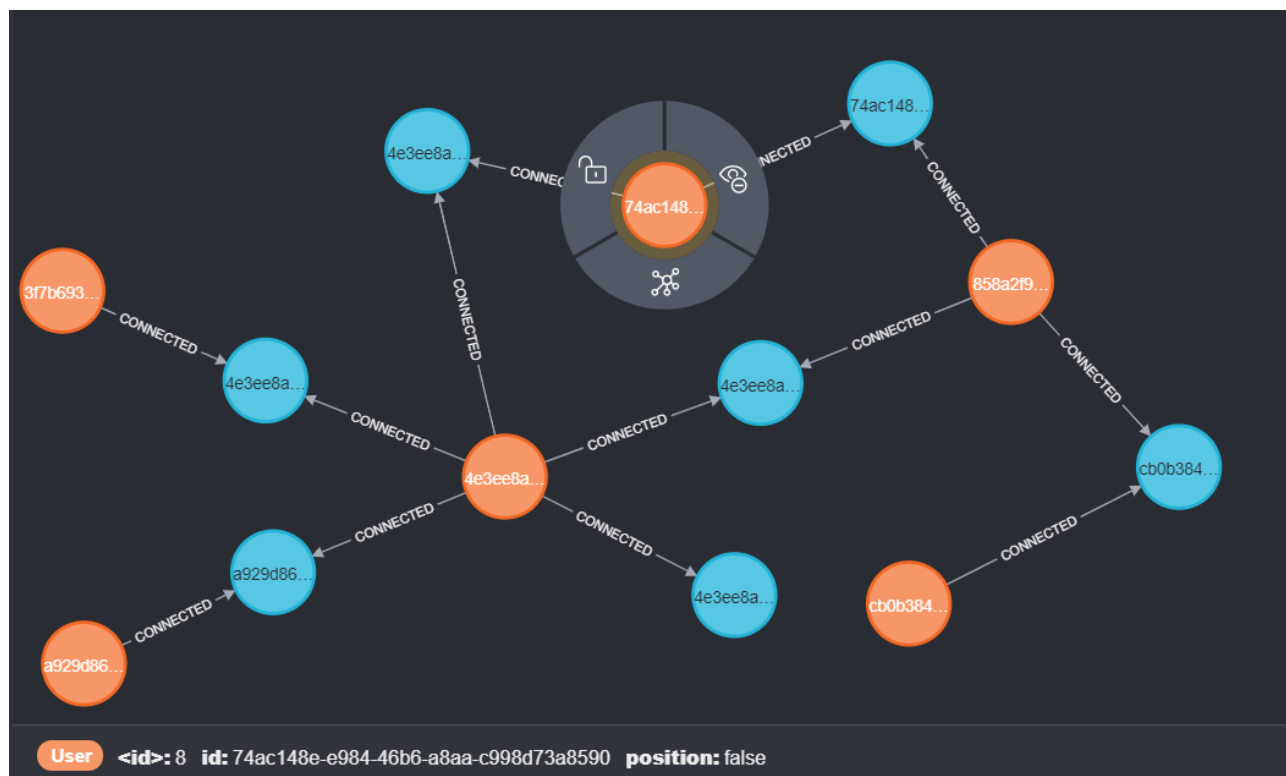
## Приклад перевірконої схеми властивостей користувача

```
src > validators > JS userValidator.js > ...
1  const joi = require("joi");
2
3  const UserRegistrationValidator = joi.object({
4    id: joi.string().max(100).required(),
5    nickname: joi.string().alphanum().min(4).max(45).required(),
6    name: joi.string().pattern(new RegExp("[A-Za-z]{3,45}")).required(),
7    surname: joi.string().pattern(new RegExp("[A-Za-z]{3,45}")).required(),
8    email: joi.string().email({ minDomainSegments: 2 }).required(),
9    password: joi.string().min(4).required(),
10   position: joi.boolean().required(),
11   about: joi.string().max(200).allow(""),
12 });
13
14 const UserAuthenticationValidator = joi.object({
15   nickname: joi.string().alphanum().min(4).max(45).required(),
16   password: joi.string().min(4).required(),
17 });
18
19 module.exports = {
20   UserRegistrationValidator: UserRegistrationValidator,
21   UserAuthenticationValidator: UserAuthenticationValidator,
22 };
23
```



## Додаток Г

## Зображення зв'язків між вузлами в Neo4j та їхнього вмісту



## Додаток Д

Приклад отримання картки нагадування про наближення зустрічі

